AMSTRAD
6128

CPC

LA BIBLE DU CPC 6128

LIVRE DATA BECKER



EDITIONS MICRO APPLICATION



EDITIONS MICRO APPLICATION

Distribué par : MICRO APPLICATION
13, Rue Sainte Cécile
75009 PARIS

et

EDITION RADIO
3, Rue de l'Eperon
75006 PARIS

ISBN : 2-86899-036-3

Traduction Française assurée par Pascal HAUSSMAN

édité par Frédérique BEAUDONNET

TABLE DES MATIERES

INTRODUCTION

La politique produit de la société AMSTRAD a de quoi surprendre. A peine le CPC 464 s'était-il établi dans le marché si disputé de l'informatique, grâce à son prix peu élevé et à ses performances remarquables, que fit son apparition sur le marché un second ordinateur, le CPC 664. Et moins de 3 mois plus tard, le CPC6128, le troisième ordinateur de la série des CPC, apparut sur le marché. Les deux successeurs du CPC 464 se distinguèrent également par un rapport qualité/prix exceptionnel.
Le caractère complet du système est encore plus frappant que pour le CPC 464. Grâce au moniteur livré avec l'appareil, pas de dispute pour savoir si on regarde Dallas ou si on utilise l'ordinateur. De même, le lecteur de disquette intégré rend inutiles les câbles de connexion et les interfaces qui faisaient de l'utilisation du lecteur de disquette un problème permanent. Votre ordinateur possède tout ce dont vous avez besoin pour pouvoir l'utiliser immédiatement.

Les possibilités de l'ordinateur sont un second point fort de ce matériel. Le Basic LOCOMOTIVE est certainement le meilleur disponible sur les ordinateurs de cette catégorie. La programmation des interruptions très souple et très facile d'emploi dont dispose ce Basic est certainement un des aspects les plus remarquables de cet ordinateur.
L'execellence du graphisme et la possibilité d'avoir un écran en 80 colonnes sans module ni coût supplémentaire est pour l'heure sans équivalent, alors que d'autres ordinateurs de la même catégorie ont déjà du mal à présenter sur l'écran 40 caractères par ligne parfaitement lisibles.
La résolution graphique de 640 points sur 200 est également unique pour cette catégorie de prix. On ne trouve de possibilités comparables que sur l'IBM PC qui est tout de même au moins cinq fois plus cher que le CPC. Les possibilités sonores du CPC sont également impressionnantes.

En ce qui concerne la vitesse, le CPC n'a pas à rougir. Le microprocesseur intégré Z80 fonctionne avec une fréquence de 4 mégaherz et il dispose d'un jeu d'instructions très puissant. Ce jeu d'instructions a été exploité au maximum par les développeurs de la machine qui ont ainsi réussi à réaliser un interpréteur Basic particulièrement rapide.

Mais les possesseurs d'un nouvel ordinateur cherchent en général très vite à obtenir plus d'informations sur leur machine. Le manuel d'utilisation du CPC, qui est par ailleurs tout à fait remarquable, ne suffit pas à répondre à l'attente de ceux qui veulent connaître leur ordinateur dans les moindres détails et notamment pour ceux, pour qui le Basic a perdu un peu de son attrait, qui en ont découvert les limites et qui souhaiteraient donc s'attaquer à la programmation en langage-machine. Il faut alors disposer d'informations qui dépassent largement le cadre du manuel d'utilisation.

Le listing de la ROM, qui figurait normalement jusqu'ici dans les ouvrages de la série "La bible du ...", est ici présenté sous une forme nouvelle, plus compacte. Nous avons préféré renoncer au listing au profit de commentaires plus complets. Vous pouvez d'ailleurs faire sortir vous-même, quand vous le voudrez votre propre listing de la ROM, grâce au désassembleur que nous vous fournissons dans cet ouvrage. A l'avenir, cette façon de commenter les systèmes d'exploitation se révèlera sans doute indispensable; en effet, les systèmes d'exploitation atteignent des dimensions de plus en plus importantes et les présenter commentés in extenso demanderait plus de pages que n'en comporte un ouvrage normal.

Les auteurs

1 LE MATERIEL (HARDWARE)

1.1 Ce que vous devez absolument savoir sur votre machine

Votre CPC contient au total six circuits essentiels, hautement intégrés. Le composant le plus important de tout ordinateur, le processeur, est sur le CPC un Z80. Les autres composants intégrés sont un video controller HD 6845, un port parallèle 8255, un sound chip AY-3-8912, le floppy controller 765 et un circuit développé spécialement pour le CPC, le Gate Array.

Le contrôleur vidéo a pour fonction de fournir tous les signaux nécessaires pour le fonctionnement du moniteur. Il adresse également la mémoire-écran, cette zone de la mémoire dans laquelle sont placés les caractères à représenter et le graphisme. Il produit également le refresh qui est nécessaire pour les Rams, sans lequel vous perdriez vite les informations stockées.

La tâche du chip sonore est définie par le nom de ce composant. Le choix des constructeurs est très bon. Le AY-3-8912 a été utilisé dans de nombreux ordinateurs parce qu'il est très polyvalent et qu'il permet des possibilités étendues d'influencer le son.

Le 8255 est le "travailleur de force" du CPC. Ses tâches sont très diverses.

Cela va du contrôle du clavier à la commande du chip sonore en passant par la commande du magnétophone, à la détermination de certaines possibilités du CPC etc...

Le floppy controller 765, en liaison avec certains circuits intégrés et ce qu'on appelle un séparateur de données, contitue l'interface avec les deux lecteurs de disquette maximum. Le floppy controller prend pratiquement en charge en totalité la commande des lecteurs de disquette. Par sa haute "intelligence", il facilite la programmation.

Le gate array est particulièrement intéressant. Ce composant commande tant de choses dans le CPC qu'on pourrait presque le qualifier de processeur auxiliaire. C'est ainsi qu'il prend en charge bon nombre des tâches concernant l'écran, telles que la représentation des différentes couleurs et les différents formats de l'écran. Tous les signaux nécessaires de synchronisation sont produits par le gate array. Les interruptions, qui interrompent le déroulement normal des programmes 300 fois par seconde, sont produites par le gate array ainsi que les signaux nécessaires à la gestion de la mémoire RAM du CPC.

1.1.1 L'organisation de la mémoire

Il y a encore 5 ans, les ordinateurs disposant de 16 K de RAM étaient considérés comme bien armés. Mais depuis l'apparition du Commodore 64, les limites de la mémoire ont été nettement repoussées. Un constructeur de micro-ordinateurs n'a de chances suffisantes de prendre une part du marché que si les magiques 64 apparaissent sur sa machine.
Mais comme les prix des composants de mémoire ont fortement chuté ces derniers mois, le fait qu'un ordinateur soit doté de 64 K octets (comme le 664) ou de 128 K octets (6128), influe peu sur son prix de revient.

D'ailleurs, il n'est pas très difficile de placer une mémoire de 64 K dans un ordinateur puisque les processeurs 8-bits, qui sont les plus répandus, peuvent tous adresser une zone de 64 kilo-octets. Le Z80 du CPC peut lui aussi adresser 64 K de mémoire sans truc particulier. Mais cela suffit normalement tout juste pour la mémoire RAM et c'est tout.

Il faut donc recourir à un procédé spécial, le bank-switching, si l'on veut pouvoir adresser plusieurs mémoires avec ce type de processeurs. Ce procédé permet en effet de choisir entre des zones de mémoire (qu'on appelle banques) ROM et RAM qui se chevauchent. Il s'agit d'un procédé qui n'utilise pas de solution matériel mais a uniquement recours à un logiciel qui organise la cohabitation des ROM et des RAM aux mêmes adresses.

Cette solution logiciel a été remarquablement mise en oeuvre par les développeurs de l'ordinateur. Sur le CPC 6128, l'adressage du bloc supplémentaire de 64 K de RAM est également réalisé grâce à ce procédé.

Le CPC 6128 présente donc l'image suivante: (remarque: sur le CPC 664, le principe est le même, si ce n'est que la banque supplémentaire de 64 K fait défaut) 64 K de RAM sont adressés directement. Parallèlement à la RAM se trouvent une moitié de la ROM dans les 16 K inférieurs (&0000 à &3FFF) et l'autre moitié de la ROM dans les 16 K supérieurs (&C000 à &FFFF), ainsi que, sur toute la zone d'adresses, la banque supplémentaire de 64 K de RAM (cette dernière n'est cependant pas adressable aussi directement).

Les 16 K inférieurs de ROM contiennent le système d'exploitation et un bloc de routines arithmétiques. Dans le système d'exploitation se trouvent toutes les routines dont le CPC a besoin pour lire par exemple un caractère tapé au clavier, pour placer un caractère ou un point sur l'écran mais c'est également le système d'exploitation qui commande le lecteur de cassette et l'interface imprimante ainsi que le son.

Dans les 16 K supérieurs se trouve l'interpréteur Basic. Ces 16 K n'ont pas de fonction spéciale. Il est possible de connecter dans cette zone jusqu'à 252 ROMs supplémentaires. C'est ainsi que les routines nécessaires pour la gestion du lecteur de disquette sont placées dans une ROM qui 'partage' cette zone avec le Basic.

La disposition de la mémoire est représentée par la figure 1.1.1.1

	RAM BANK Ø	RAM BANK 1	ROM	ROM	
FFFF					
	Block 3	Block 3	BASIC	AMSDOS	max. 251 ext. ROMs
CØØØ					
	Block 2	Block 2			
8ØØØ					
	Block 1	Block 1			
4ØØØ					
	Block Ø	Block Ø	Betriebs-System		
ØØØØ					

1.1.1.1 Organisation de la mémoire du CPC

1.1.2 Extension d'instructions à travers RST

Etant donné ce mode de gestion de la mémoire, on peut cependant se demander comment peut se faire l'accès aux ROMs ou aux RAMs situées dans les même zones. Pour éviter aux utilisateurs le travail de programmation assez considérable que nécessiteraient normalement ces tâches, les programmeurs du système d'exploitation ont eu une riche idée. Grâce à des programmes spéciaux et à une utilisation habile des instructions RESTART du Z80, ils ont pratiquement abouti à faire des restarts RST1 à RST5 une extension du jeu d'instructions du Z80. Ces RSTs peuvent être employés comme des JPs ou des CALLs ordinaires. Certains RSTs réclament toutefois une adresse sur 3 octets. Le troisième octet, supplémentaire, détermine dans quelle ROM le JP ou le CALL doit aller.

LOW JUMP RST 1

Cette instruction Restart permet d'appeler une routine du système d'exploitation ou de la RAM située dans la même zone d'adresses. L'instruction RST doit être suivie immédiatement par l'adresse de la routine à appeler. Comme 14 bits suffisent pour définir une adresse comprise entre 0 et &3FFF, les deux bits supérieurs restants sont utilisés pour sélectionner la ROM ou la Ram:

Bit 14=0	Sélection du système d'exploitation
Bit 14=1	Sélection de la Ram
Bit 15=0	Sélection de la ROM Basic
Bit 15=1	Sélection de la Ram

Un appel de la routine système pourrait donc se présenter ainsi:

```
RST 1
DW &1410+&8000
```

Le bit 15 mis sélectionne la RAM dans la zone de &C000 à &FFFF, alors que le bit 14 annulé appelle le système d'exploitation.

Le code à l'adresse 8 est constitué uniquement par un saut à l'adresse &B98A.

SIDE CALL RST 2

Cette instruction Restart permet d'appeler une routine d'une ROM d'extension. Cette instruction est utilisée lorsqu'un programme sous forme d'un module de ROM nécessite plus de 16 kilo-octets et ne peut pas tenir dans un seul module d'extension. Le SIDE CALL permet alors d'appeler une routine se trouvant dans la seconde, la troisième ou la quatrième ROM appartenant au programme, sans qu'il soit pour cela nécessaire de connaître le numéro absolu de la ROM qu'il s'agit d'appeler ainsi. L'instruction RST 2 doit être suivie de l'adresse de la routine - &C000, c'est-à-dire de l'adresse relative par rapport au début de la ROM. Les deux bits supérieurs servent à sélectionner l'une des quatre ROMs différentes utilisées.

Le code à l'adresse &0010 est constitué uniquement par un saut à l'adresse &BA1D.

FAR CALL RST 3

Cette instruction Restart permet d'appeler une routine n'importe où en ROM ou en RAM. L'instruction RST 3 doit être suivie de l'adresse sur deux octets d'un bloc de paramètres composé de trois octets. Les deux premiers de ces octets-paramètres contiennent l'adresse de la routine qui doit être appelée et le troisième octet doit contenir l'état ROM/RAM souhaité. Les valeurs de 0 à 251 permettent d'appeler une ROM supplémentaire et les quatre valeurs restantes ont la fonction suivante:

Valeur	&0000-&3FFF	&C000-&FFFF
252	Système d'exploitation	Basic
253	Système d'exploitation	RAM
254	RAM	Basic
255	RAM	RAM

Le code à l'adresse &0018 est constitué uniquement par un saut à l'adresse &B9C7.

RAM LAM RST 4

Cette instruction Restart permet de lire à partir d'un programme en langage-machine le contenu de la RAM, quel que soit l'état de la ROM choisi. L'instruction RST 4 remplace alors l'instruction

LD A,(HL)

HL doit donc contenir l'adresse de la case mémoire dont le contenu doit être lu. Le code à l'adresse &0020 est constitué uniquement par un saut à l'adresse &BAD6.

FIRM JUMP RST 5

Cette instruction Restart permet de sauter à une routine du système d'exploitation. L'adresse doit être placée immédiatement à la suite de l'instruction RST 5. La ROM du système d'exploitation est sélectionnée avant le saut à la routine puis elle est déconnectée aprés le retour. Le code à l'adresse &0028 est constitué uniquement par un saut à l'adresse &BA35.

1.2 Le processeur Z80

Le début des années 70 a connu le triomphe des microprocesseurs. La société INTEL a pu se tailler avec le processeur 8080 une part significative du marché parce qu'au moment où elle le lança sur le marché, il n'avait pratiquement pas de concurrent dans cette catégorie. C'est bien ce qui frappe quand on examine de plus près les données techniques de ce processeur. Le 8080 avait en effet besoin de trois tensions de courant différentes et de deux circuits intégrés supplémentaires pour la production des signaux de commande et de synchronisation.

La société ZILOG a développé le Z80 dans les années 74/75. Mais au lieu de développer un processeur radicalement nouveau, on s'en tint à la conception du 8080 qui avait rencontré un tel succés. C'est pourquoi le Z80 est compatible avec le 8080 (mais non pas l'inverse). C'est-à-dire que tous les programmes écrits pour un 8080 tournent aussi sur un Z80.

Cependant toutes les particularités considérées comme néfastes du 8080 furent éliminées et le jeu d'instructions fut largement étendu. Le Z80 ne nécessite d'autre part qu'une tension de +5Volt et il n'a pas besoin de circuits intégrés externes pour produire les signaux de commande.

Mais examinons en style télégraphique les données techniques de ce processeur, avant que nous n'entrions plus dans le détail de ses caractéristiques:

Processeur 8-bits de technologie NMOS
Bus d'adresses 16-bits
Alimentation unique 5 Volt
Fréquence simple
Compatible TTL
Fréquence d'horloge de 2.5, 4, 6 ou même 8 MHz
Compatibilité logiciel avec le 8080
Double jeu de registres plus deux registres d'index
Entrée d'interruptions non-masquable
Entrée d'interruptions masquables avec trois modes de

travail
Refresh automatique de RAMs dynamiques
Circuits intégrés périphériques du 8080 directement connectables

Ces données techniques ainsi qu'un grande masse de logiciels disponibles ont fait du Z80 l'un des processeurs 8-bits les plus répandus. Dans le domaine des ordinateurs familiaux et personnels, seul le 6502 a obtenu une diffusion comparable.

A 11	1	A 10
A 12		A 9
A 13		A 8
A 14		A 7
A 15		A 6
Ø		A 5
D 4		A 4
D 3		A 3
D 5		A 2
D 6		A 1
+5V		A 0
D 2		GND
D 7		RFSH*
D 0		M1*
D 1		RESET*
INT*		BUSRQ*
NMI*		WAIT*
HALT*		BUSAK*
MREQ*		WR*
IORQ*		RD*

1.2.1.1 PINOUT du Z80

1.2.1 Les connexions du Z80

Après ce bref aperçu sur les possibilités du Z80, intéressons-nous maintenant à l'affectation des 40 pins de connexion du Z80.

Les points de connexion du Z80 peuvent être répartis entre les 4 groupes bus de données, bus d'adresses, bus de commande et canaux de transmission.

Bus d'adresses

A0 - A15 : Lignes d'adresses; ces connexions permettent d'appeler une case mémoire dans la zone adressable qui comprend 65536 cases mémoire. Dans le traitement des instructions d'entrée-sortie, les 8 bits inférieurs de l'adresse sont utilisés pour sortir l'adresse d'entrée-sortie correspondante. 256 ports différents sont ainsi possibles. Avec certaines limites tenant au jeu d'instructions, ce sont même 65536 ports qui peuvent être adressés. Les 16 canaux d'adresse sont alors utilisés pour constituer l'adresse du port. Nous reviendrons plus tard sur ce cas particulier.

Bus de données

D0 - D7 : Lignes de données; ces canaux bidirectionnels transmettent les données venant du processeur ou allant vers le processeur. Elles font le lien entre le processeur et la case mémoire ou l'adresse de port choisies à travers le bus d'adresses.

Bus de commande

M1* : Machine Cycle One; ce signal de commande indique que le processeur lit le code d'instruction sur le bus de données. L'étoile signifie par ailleurs pour ce signal et pour les signaux suivants, qu'il s'agit d'un signal actif avec low.

MREQ* : Memory REQuest*, ce signal de sortie indique par un low que le processeur entreprend un accès en lecture ou écriture à une adresse de la mémoire et que l'adresse sur le bus d'adresses est valable.

IORQ* : Input/Output ReQuest*, ce signal de sortie indique par un low que le processeur entreprend un accès en lecture ou écriture à une adresse de port et que l'adresse de port sur le bus d'adresses est valable.

RD* : ReaD*, ce signal de sortie indique par un low que le processeur veut lire des données dans une case mémoire ou dans une adresse de port. L'utilisation conjointe avec MREQ* ou IORQ permet de distinguer entre la lecture de la mémoire ou d'un port.

WR* : WRite*, ce signal indique, lors d'accès en écriture du processeur à la mémoire ou aux adresses de port, que les données figurant sur le bus de données sont valables. Ici aussi, l'utilisation conjointe de WR* avec MREQ* ou IORQ* permet de distinguer si les données doivent être écrites dans la mémoire ou dans une adresse de port.

RESET* : Lorsque ce signal d'entrée passe à low, le compteur de programme reçoit la valeur &0000, les interruptions sont interdites et le mode d'interruption 0 est activé. Dès que ce signal d'entrée redevient high, le processeur commence l'exécution du programme à partir de l'adresse &0000.

NMI* : Non Maskable Interrupt*, ce signal d'entrée provoque toujours par un double signal high-low une interruption du programme exécuté par le processeur. Les valeurs placées en &0066 et &0067 sont alors chargées dans le compteur de programme et le programme se poursuit à partir de cet endroit.

IRQ* : Interrupt ReQuest*, ce signal d'entrée peut provoquer par un low une interruption du programme exécuté par le processeur, à condition que ce type d'interruptions soit autorisé par instruction. Les effets dépendent du type d'interruption et seront évoqués plus tard. IRQ* est, au contraire de NMI*, un signal statique qui doit persister jusqu'à ce que la demande d'interruption ait été prise en compte.

WAIT* : Ce signal permet d'adapter l'accès en lecture ou en écriture du Z80 à des mémoires plus lentes ou à des conditions spéciales du systéme.

BUSRQ* : BUSReQuest*; losque ce signal d'entrée passe à low, les canaux de données et d'adresses ainsi que tous les canaux de commande de sortie deviendront high après le traitement de l'instruction actuelle et le signal BUSAK* deviendra low. Maintenant, un second processeur pourrait prendre en charge l'accès à la mémoire et aux éléments périphériques; ce signal est cependant essentiellement utilisé pour le DMA (DMA= Direct Memory Access, transfert de données très rapide en contournant le processeur).

BUSAK* : BUSAKnowledge*, est le signal de sortie correspondant à BUSRQ*. Un low indique au DMA controller ou au second processeur que tous les signaux de commande et de bus sont high et qu'un accès est maintenant possible.

HALT* : Ce signal de sortie devient low après que le processeur ait exécuté l'instruction en langage-machine HALT. Après cette instruction, le processeur ne fait plus rien d'autre que d'exécuter des NOPs pour assurer le Refresh. Seule une interruption peut à nouveau le "réveiller".

RFSH* : ReFreSH*, ce signal de sortie indique que les sept canaux d'adresses inférieurs contiennent une adresse de Refresh valable. Comme le processeur n'a besoin du bus d'adresses et de données qu'à certains moments, le bus d'adresses peut être utilisé le reste du temps pour rafraîchir les RAMs dynamiques, sans qu'une électronique coûteuse ou des routines spéciales de rafraîchissement soient pour cela nécessaires.

Horloge et alimentation électrique

0 :Phi : Le signal d'entrée phi sert d'horloge pour le processeur. Comme le Z80 est un circuit intégré statique, la fréquence d'horloge peut varier entre 0 Hertz et la fréquence maximale indiquée. La forme du signal d'horloge doit cependant répondre à certaines exigences. Le durée low de ce signal ne doit pas dépasser 2 microsecondes. Cette valeur n'a d'ailleurs qu'un intérêt théorique, puisqu'on essaiera évidemment toujours de fournir au processeur une fréquence d'horloge la plus élevée possible, de façon à obtenir une exécution rapide du programme.

GND : Branchement à la masse du processeur.

Vcc : C'est par cette connexion que le Z80 reçoit son alimentation en courant électrique continu de 5 Volts et environ 150 à 200 milliampères.

1.2.2 LA STRUCTURE DES REGISTRES DU Z80

Comme nous l'avons indiqué au début, le Z80 a été construit de telle façon que les programmes du 8080 puissent être repris sans problème. Mais le Z80 dispose d'un nombre de registres nettement supérieur.

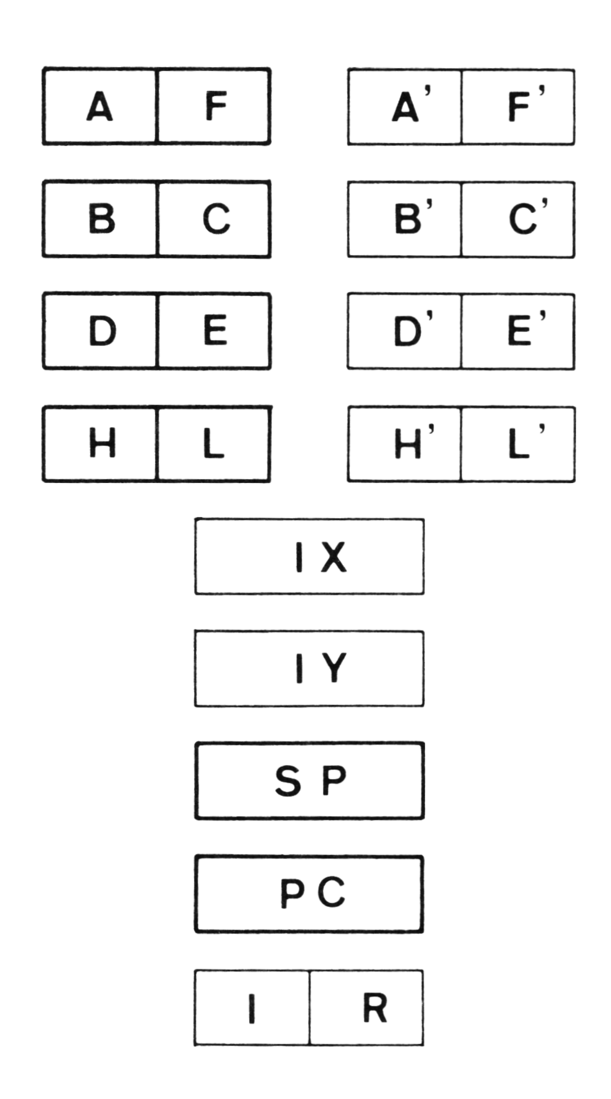
Mais qu'est-ce donc qu'un registre?

Un registre n'est rien d'autre qu'une mémoire de lecture/écriture sur le chip du processeur. Chaque processeur doit disposer d'un minimum de registres. Dans ces cases de mémoire, les données peuvent être placées, ainsi que les résultats d'instructions arithmétiques et logiques. D'autres registres ont des fonctions spéciales, telles que la gestion de la pile, ou sont utilisés comme compteur de programme.

Comme les opérations telles qu'un transfert de données entre deux registres ou l'addition des contenus de deux registres ne peuvent se faire à travers le bus de données, de telles opérations peuvent être exécutées beaucoup plus rapidement que lorsque les valeurs nécessaires doivent être recherchées dans des cases de mémoire externes.

On peut donc dire en règle générale que les processeurs disposant d'une mémoire interne plus importante sont supérieurs aux processeurs disposant de peu de registres pour le traitement des mêmes programmes car le transfert de données est toujours plus rapide à l'intérieur du processeur qu'entre le processeur et les cases de mémoire externes.

Le Z80 dispose de 22 registres au total, 18 registres de 8 bits et 4 registres de 16 bits. La figure 1.2.2.1 montre la disposition de ces registres.



1.2.2.1 Jeu de registres Z80

Dans cette figure, certains registres sont marqués par un cadre plus épais. Ces registres existent également sur le 8080.
Vous voyez également que la plupart des registres 8 bits apparaissent en double exemplaire. Il s'agit des registres A, F, B, C, D, E, H et L. Le programmeur peut choisir entre deux jeux de registres.

Nous ne parlerons à l'avenir que d'un seul jeu de registres, d'autant que le programmeur du CPC ne dispose en fait, à moins de recourir à certaines astuces particulières, que d'un seul jeu de registres. Le jeu de registres alternatif est utilisé par le système d'exploitation pour la gestion des interruptions. Mais notez bien que toutes les tâches d'un jeu de registres peuvent également être prises en charge par le jeu de registres alternatif, si celui-ci n'est pas employé pour des opérations spécifiques.

Les registres B à L sont les registres 8 bits normalement disponibles, alors que les registres A et F répondent à des tâches particulières.

Le registre A est généralement qualifié d'accumulateur. C'est dans l'accumulateur qu'on obtient le résultat de toutes les opérations arithmétiques et logiques sur 8 bits. Pour ces opérations, un opérande doit d'autre part être placé dans l'accumulateur. Pour additionner par exemple deux octets, il faut placer un opérande dans l'accumulateur alors que le second opérande peut être placé dans un autre registre du processeur ou dans une case de la mémoire externe. Après l'addition, le résultat se trouve dans l'accumulateur.
Comme, lors de telles opérations, le résultat peut être supérieur à la valeur maximale qui peut être exprimée avec 8 bits (255+255=510), un bit supplémentaire est nécessaire pour représenter le résultat correctement. C'est le registre F qui remplit cette fonction. Le registre F, généralement qualifié de registre flag est divisé en ses différents bits. Un de ces bits a entre autre pour fonction de conserver une éventuelle retenue (carry en anglais) résultant de telles additions. Les autres bits indiquent si le résultat d'opérations de calcul ou de comparaisons est nul, etc...

Les registres B à L ne peuvent toutefois pas uniquement être appelés séparément. B et C, D et E ainsi que H et L peuvent être regroupés en registres 16 bits. Ces registres 16 bits reçoivent alors naturellement les noms BC, DE et HL. Les registres doubles conviennent parfaitement à l'adressage de tableaux ainsi qu'au transfert et à la recherche de blocs de données.

Le registre double HL a une signification particulière. Comme le Z80 dispose d'instructions d'addition et de soustraction sur 16 bits, le registre HL fait office, pour de telles instructions, d'accumulateur 16 bits.

Les registres PC, SP, IX et IY ne travaillent qu'avec des valeurs 16 bits (remarque: les spécialistes savent qu'il est également possible de manipuler les registres d'index octet par octet mais nous ne considèrerons IX et IY que comme de purs registres 16 bits).

Le registre PC est le compteur de programme (Programm Counter). Le contenu du PC est placé sur le bus d'adresse comme adresse pour les mémoires externes. Avec chaque instruction, le PC est incrémenté (augmenté de 1) automatiquement. Pour les instructions sur plusieurs octets, le PC est automatiquement augmenté de la valeur correspondant à ce nombre d'octets. Si des sauts doivent se produire à l'intérieur d'un programme, la nouvelle adresse du programme est automatiquement chargée dans le PC et le processeur continue l'exécution à partir de cette adresse.

Le registre SP est le pointeur de pile (Stack Pointer). La pile est utilisée lorsque des sous-programmes sont appelés. Dans ce cas en effet, l'adresse de retour est automatiquement placée sur la pile puis rechargée dans le PC après exécution du sous-programme.

Les deux registres 16 bits IX et IY permettent, grâce à des instructions spéciales, un travail particulièrement efficace avec les tableaux.

Il ne reste plus que les registres I et R. Le registre I ou registre d'interruption (Interrupt Register) est utilisé en liaison avec le mode d'interruption spécial IM3. Dans ce mode d'interruption, l'élément produisant l'interruption doit fournir, à la demande du processeur, une valeur 8 bits. Cette valeur comme low byte et le contenu du registre I comme high byte constituent l'adresse de la routine d'interruption.

Le registre R ou Refresh Register est utilisé en liaison avec le Refresh que le Z80 exécute automatiquement. Chaque fois qu'une instruction a été retirée, les sept bits inférieurs de ce registre sont automatiquement incrémentés. Le huitième bit reste toujours à 1 ou à 0, suivant sa programmation.

Les registres I et R ne sont pas utilisés sur le CPC. Cependant, comme la valeur du registre R se modifie sans cesse, celui-ci peut être utilisé comme générateur de hasard.

1.2.3 Particularités du Z80 du CPC

Les nombreuses possibilités du Z80 laissent une grande marge de manoeuvre aux concepteurs de matériel ou de logiciel dans la construction d'un ordinateur. Cette CPU (unité centrale) peut être utilisée avec la même efficacité dans des systèmes très réduits ainsi que dans des machines aussi puissantes que le CPC.

Les développeurs du CPC se sont ingéniés à obtenir un maximum de puissance avec un minimum de composants. D'où certaines particularités qu'il est nécessaire de connaître pour pouvoir programmer et utiliser efficacement cette machine, particulièrement en langage-machine. Ce sont ces particularités que nous allons maintenant étudier.

Tout d'abord la gestion des interruptions du CPC.
La seule source d'interruptions du CPC est le gate array, ce composant fantastique qui contribue de façon décisive à la puissance de cet ordinateur. Toutes les 3,3 millisecondes, soit 300 fois par seconde, le gate array produit une brève impulsion qu'il

place sur l'entrée IRQ* du Z80.
L'entrée NMI* du processeur n'est pas utilisée et est disponible sur le connecteur d'extensions pour des extensions éventuelles.
La fréquence du signal d'interruptions est obtenue, à partir du signal H-Sync du CRTC 6845, au moyen d'un diviseur de fréquence. L'impulsion H-sync qui apparaît environ toutes les 65 microsecondes est ici divisée par 52.

Comme le Z80 fonctionne sur le CPC en mode d'interruption IM1, chaque interruption IRQ identifiée provoque un RST7 ou encore un CALL &0038. Le processeur interrompt immédiatement le programme en cours, place l'état actuel du PC sur la pile et saute à l'adresse &0038. Ici figure, sur le CPC, un saut à l'adresse où se trouve la routine d'interruption proprement dite. Comme l'endroit où s'est produit l'interruption est enregistré sur la pile, le programme interrompu peut être repris une fois terminée la routine d'interruption.

Comme l'entrée IRQ* du processeur se trouve également sur le connecteur d'extension, on peut bien sûr se demander comment une interruption par le gate array peut être distinguée d'une interruption externe. Les développeurs du CPC ont eu ici recours à une astuce. A l'intérieur de la routine d'interruption, l'interruption est à nouveau autorisée un court instant. Comme l'impulsion produite par le gate array ne dure pas plus de 5 microsecondes, cette autorisation de l'interruption n'a aucun effet, puisque l'impulsion est terminée depuis longtemps. Par contre, les sources externes d'interruption ne mettent fin à l'émission de leur signal que sur instruction expresse du processeur. Lorsqu'il y a une interruption externe, la routine d'interruption est donc elle-même interrompue. Ce cas peut être identifié et traité d'une manière spéciale. C'est ainsi que sont rendues également possibles les sources d'interruptions externes. La seule condition qu'elle doivent remplir, c'est une impulsion suffisamment longue.

Le second cas particulier qui doit être pris en compte, c'est la possibilité limitée d'utiliser les instructions de port.

En liaison avec le signal IORQ* (Input/Output ReQuest), le Z80 peut adresser un maximum de 256 ports différents, de façon analogue à l'adressage de cases mémoire. Pour cela, l'adresse du port souhaité est placée dans les 8 bits inférieurs d'adresse A0 à A7. Ces ports sont essentiellement utilisés pour connecter des éléments périphériques.

Sur d'autres processeurs qui ne connaissent pas l'adressage de port, le concepteur est toujours tenté d'adresser les éléments périphériques comme des cases mémoire. Ce procédé est appelé Memory Mapped et il présente l'inconvénient de réduire la zone d'adresses disponible pour la RAM.

Pour l'utilisation de l'adressage de port, le Z80 fournit le groupe très puissant des instructions IN et OUT. Si l'on étudie plus attentivement les instructions de ce groupe, on trouve dans les instructions IN(C),r et OUT(C),r une possibilité élégante d'adresser plus que les 256 ports normalement prévus. Dans ces instructions, l'état des 8 bits d'adresse inférieurs est déterminé par le contenu du registre C mais le contenu de B est en outre placé dans les bits d'adresse A8 à A15. C'est ainsi 65536 adresses de ports qui sont disponibles. C'est justement cette caractéristique du Z80 que les concepteurs du CPC ont utilisée. Tous les circuits intégrés périphériques sont sélectionnés au moyen des bits d'adresse A8 à A15.

De telles astuces ont malheureusement souvent un inconvénient. En l'occurence l'inconvénient réside dans une nette limitation du jeu d'instructions du Z80. Aucune des autres instructions I/O du Z80 ne peut plus être utilisée. Ceci vaut notamment pour les instruction I/O avec automatisme de boucle. Ces instructions utilisent le registre B comme compteur et ne peuvent donc pas 'fournir' l'octet fort de l'adresse de port. C'est en particulier le cas des instructions INI, INIR, IND et INDR ainsi que OUTI, OTIR, OUTD et OTDR.

L'utilisation des cycles wait constitue une troisième particularité du CPC.

La nécessité de cette connexion du Z80 remonte à l'époque où les circuits intégrés de mémoire disponibles se la coulaient encore douce. Les premières EPROMs notamment n'étaient pas en mesure de préparer les données, après réception de l'adresse, avant un délai de quelques microsecondes.

Pour faire fonctionner le Z80 avec de tels 'paresseux', il fallait attendre un certain temps. Ce délai peut être produit par le signal WAIT*. Lors de chaque signal négatif sur l'entrée de l'horloge, le processeur examine l'état de la connexion WAIT*. Si cette connexion est à 0 Volt, le Z80 exécute ce que l'on appelle un cycle d'attente de la durée d'un mouvement d'horloge. Une fois écoulé le signal d'horloge, donc avec le signal négatif, l'état du canal WAIT* est à nouveau examiné, etc... L'utilisation de ce signal sur le CPC n'a cependant aucun rapport avec les circuits intégrés de mémoire utilisés. Ils sont tous suffisamment rapides pour un Z80 d'une fréquence de 4 MHz. La raison de l'utilisation de cette connexion est la nécessaire synchronisation entre processeur et contrôleur vidéo. Comme les deux circuits intégrés peuvent accéder à la mémoire, il faut contrôler de qui c'est le tour à un moment donné. Le contrôleur vidéo est d'ailleurs toujours prioritaire car sinon l'affichage sur le moniteur pourrait être sérieusement endommagé. Pour obtenir cette synchronisation, un signal WAIT* est produit pour le processeur tous les 4 mouvements d'horloge. Bien que le processeur fonctionne à 4 MHz (Méga Hertz= millions de vibrations par seconde), du fait des cycles d'attente, la fréquence de travail effective est d'environ 3,3 MHz.

Les signaux BUSRQ* et BUSAK*, les signaux de commande du DMA ne sont pas utilisés sur le CPC. Ils sont cependant placés sur le connecteur d'extension et sont donc disponibles pour des extensions externes.

Le signal HALT*, qui n'est pas non plus utilisé sur le CPC est également disponible sur le connecteur d'extension.

1.3 Le gate array, le coordinateur du système

Presque tous les composants du CPC se trouvent couramment dans le commerce, dans n'importe quel magasin d'électronique bien approvisionné. Les seules exceptions sont la ROM et le gate array qui est désigné dans le schéma technique sous le nom de IC116. C'est ce dernier circuit intégré qui nous occupera dans cette section.

Ce circuit intégré à 40 pôles a été développé spécialement pour le CPC et il remplit plusieurs fonctions importantes. Si l'on voulait reconstituer toutes les fonctions intégrées avec des portes logiques TTL, le nombre de circuits intégrés ferait vite plus que doubler.

Les fonctions du gate array sont entre autres les suivantes:

Production de toutes les fréquences d'horloge nécessaires
Production des signaux pour l'exploitation de la RAM dynamique
Commande des accès à la RAM
Connexion et déconnexion de la ROM sur la zone de mémoire
Production des signaux vidéo
Production des informations RVB pour le moniteur couleur
Commande du mode d'écran
Stockage des couleurs d'encre
Production de l'impulsion d'interruption

Il n'y a malheureusement que très peu d'informations disponibles sur ce circuit intégré très intèressant. Il est impossible d'obtenir une description technique de ce circuit intégré dont la vie interne est visiblement considérée par le constructeur comme un secret de fabrication.

Mais nos efforts et tentatives de découvrir le fonctionnement de ce circuit intégré de la façon la plus détaillée possible ont débouché sur un réel succés et nous ne voulons pas vous cacher les résultats auxquels nous avons abouti.

1.3.1 L'affectation des pôles de connexion du gate array

Avant d'en venir aux fonctions des différentes connexions du gate array, nous vous devons quelques explications. Il existe maintenant au moins trois versions différentes du gate array. Sur le CPC 464, le premier ordinateur CPC, ce circuit intégré portait la référence 40007. Ce circuit intégré s'échauffait tellement pendant le fonctionnement le l'appareil, qu'il a été nécessaire de le refroidir en y collant une feuille d'aluminium. Cette situation ne pouvait durer à la longue car, même avec un refroidissement supplémentaire, le circuit intégré pouvait être détruit par échauffement. Sur la CPC 664, c'est l'IC 40008 qui fut employé. Grâce à des modifications dans sa structure interne, l'énergie perdue convertie en chaleur put être réduite. Cette version chauffait beaucoup moins. Le CPC 6128, enfin, utilise l'IC 400010. Sur ce circuit intégré, l'ordre des connexions a été modifié. Il est cependant également possible d'utiliser les anciennes versions du gate array dans le CPC 6128. La place nécessaire a été laissée sur la plaque des composants.

Dans la description des connexions, nous nous sommes limité à la version utilisée dans le CPC 6128. Les numéros de connexion indiqués entre parenthèses se rapportent aux versions plus anciennes du GA, utilisées sur le 664 et le 464.

Le signal qui détermine tout sur le CPC est le signal quarz d'une fréquence de 16 MHz qui se trouve sur le pin 24 (pin 8) (XTAL). Le IC125, un circuit intégré TTL du type 7400, constitue avec deux de ses quatre portes logiques une commutation d'oscillateur typique. Ce signal constitue pratiquement le battement cardiaque du CPC.

La fréquence d'entrée divisée par quatre est disponible pour le processeur, sous la forme d'un signal d'horloge de 4 MHz sur le pin 19 (pin 39) comme fréquence Phi.

Une nouvelle division par quatre donne une fréquence de 1 MHz. Ce signal est fourni sur le pin 14 (pin 1) du gate array.

Le signal de 1 MHz a deux emplois. C'est tout d'abord le signal d'horloge pour le chip sonore et il contribue ensuite à déterminer si le processeur ou le CRTC peut adresser la RAM. S'il y a un low, les canaux d'adresse du processeur sont commutés sur la RAM à travers les circuits intégrés multiplexeurs IC 74LS153.

Comme par ailleurs la commande de la RAM sur le CPC n'est pas tout à fait évidente, vous trouverez une description détaillée des signaux de commande de la RAM dans un prochain chapitre.

Left	Right
CPU ADDR* (1)	MA0/CCLK
READY	Ø
CAS*	Vcc1
244EN*	RESET*
MWE*	R
CAS ADDR*	GND
RAS*	G
XTAL	Vcc2
Vcc2	B
INTERRUPT*	D 7
SYNC*	D 6
ROMEN*	D 5
RAMRD*	D 4
HSYNC	D 3
VSYNC	D 2
IORQ*	D 1
M1*	D 0
MREQ*	DISPEN
RD*	Vcc1
A 15	A 14

1.3.1.1 PINOUT des GATE ARRAY 40007 & 40008

D 5	1	D 4
D 6		D 3
D 7		D 2
CCLK		D 1
SYNC*		V_{SS}
V_{DD}		D Ø
RESET*		RAS*
B-Ausgang		MWE*
DISPEN		INT*
C-Ausgang		CAS ADDR*
HSYNC		A 14
R-Ausgang		RAM RD*
YSYNC		A 15
CPU*		ROMEN*
V_{SS}		V_{SS}
CAS*		V_{DD}
MREQ		CK 16
IORQ*		244 EN*
PHI		READY
NMI		RD*

1.3.1.2 PINOUT du GATE ARRAY 40010

Comme les composants de mémoire ne disposent que de 8 canaux d'adresse, l'adresse totale de 16 bits doit être multiplexée, c'est-à-dire placée sur les entrées avec un décalage dans le temps. Cette commande dans le temps est obtenue avec les signaux CAS ADDR* pin 31 (pin 6), CAS* pin 16 (pin 3) et RAS* pin 34 (pin 7). Ces signaux RAS* et CAS* sont placés directement vers les RAMs, le signal CAS ADDR* est conduit vers les multiplexeurs que nous avons déjà évoqués.

Le signal MA0/CCLK sur le pin 4 (pin 40) du gate array a également une fréquence de 1 MHz. Ce signal est par ailleurs déphasé par rapport au signal CPU ADDR*, c'est-à-dire que les deux fréquences sont high à des moments différents. MA0/CCLK a également une double fonction. Il constitue d'une part le signal d'horloge pour le CRTC qui tire tous les autres signaux de ce signal; d'autre part il est placé comme bit d'adresse auxiliaire sur un des quatre multiplexeurs d'adresse. La fonction de ce bit d'adresse auxiliaire sera également évoquée plus tard plus précisément, à propos de la commande de la RAM.

Le gate array produit encore sur le pin 29 (pin 13) le signal RAMRD*. Cette connexion devient low, lorsque le processeur, après avoir fourni une adresse, veut lire des données dans la RAM et qu'il l'indique au gate array par son signal RD* sur le pin 21 (pin 19). Comme la ROM et la RAM se chevauchent sur de grandes zones, le signal RD* du processeur ne peut être utilisé directement. Si des données doivent être lues dans la ROM, le signal RAMRD* reste high et les sorties du DATA LATCH/BUFFER 74LS373 (un buffer est une mémoire provisoire) deviennent high. Dans ces moments, aucune information ne peut passer de la RAM sur le bus de données, bien que l'adresse de la mémoire soit également parvenue à la RAM et que celle-ci tienne un octet prêt dans ses sorties.

En plus du RAMRD*, le signal READY du pin 22 (pin 2) du gate array est placé sur l'IC 74LS373. Ce signal produit sur le processeur le signal pour l'insertion des cycles d'attente. La liaison supplémentaire entre le READY et le LATCH/BUFFER permet d'obtenir que l'information sur le bus de données du processeur ne se modifie pas pendant les cycles d'attente.

Le 74LS373 stocke, après envoi d'un high sur le pin 11, l'information en sortie actuelle, jusqu'à ce que ce pôle devienne low. Le circuit intégré se comporte ensuite comme un simple buffer, c'est-à-dire que les sorties suivent immédiatement les modifications des entrées.

Le signal ROMEN* sur le pin 27 (pin 12) du gate array devient low lorsque le processeur veut lire des données dans la ROM. La ROM intégrée de 32 K du CPC occupe les zones d'adresses &0000 à &3FFF et &C000 à &FFFF. Cette ROM peut donc être appelée en deux moitiés distinctes. Dans les zones de mémoire où RAM et ROM se chevauchent, il faut indiquer au gate array le choix fait avec une instruction OUT. Il est ainsi tout à fait possible de n'activer qu'une moitié de la ROM.

Conformément à la configuration de la mémoire choisie, le gate array décode l'état des canaux d'adresse A14 et A15. Suivant la mémoire demandée c'est le signal RAMRD* ou ROMEN* qui sera activé lors de la lecture.

Une instruction d'écriture du processeur va toujours vers la RAM, indépendamment de la configuration de la mémoire choisie. Le gate array produit à cet effet le signal MWE*.

Outre la fonction décrite, les canaux d'adresse A14 et A15 sur les pins 28 (20) et 30 (21) sont encore utilisés dans un autre but. Le gate array a une adresse de port qui est utilisée pour programmer les différentes possibilités du gate array. L'adresse de port est &7F00 et elle est décodée sur le pin 17 (pin 18), à travers les canaux d'adresse (A14 High, A15 Low) et le signal IORQ*.

Comme le bus de données du Z80 n'est pas directement relié aux canaux de données D0 à D7 du gate array, le GA (gate array) met le pôle 244EN* sur low lorsque l'adresse de port &7F00 est identifiée de la façon que nous avons indiquée. Les sorties du 74LS244, un buffer de bus de données, sont ainsi libérées et l'octet fourni par le Z80 peut être écrit dans le GA.

Mais le signal IORQ* a lui aussi une double signification pour le GA. Le Z80 a en effet la particularité, lorsqu'il identifie une interruption, de mettre simultanément à low les signaux IORQ* et M1*. Cette situation est identifiée par le GA et l'impulsion d'interruption est immédiatement annulée. Si, par contre, le traitement de l'IRQ a été interdit par l'instruction DI, Disable Interrupt, le pôle 10 du GA reste low, jusqu'à ce que l'IRQ soit à nouveau autorisé. Dès que l'IRQ est à nouveau autorisé par l'instruction EI, Enable Interrupt, l'interruption présente est identifiée et la sortie d'interruption redevient high.

Le signal d'interruption sur le pin 32 (pin 10) est produit par une chaîne de division programmable du GA. Cette chaîne de division est alimentée par le signal HSYNC du CRTC et elle divise la fréquence existante par 52. Comme l'impulsion HSYNC se produit environ toutes les 65 microsecondes, l'intervalle entre deux impulsions d'interruption est de 3,3 millisecondes. Les impulsions sont couplées avec le signal VSYNC du CRTC. La durée du VSYNC est programmée dans le CRTC à environ 500 microsecondes. Après environ 125 microsecondes apparaît l'interruption, de sorte que la routine d'interruption a encore environ 375 microsecondes pour examiner sur le bit 0 du port B du 8255 s'il y a un VSYNC. Ce signal est utilisé comme horloge dans différentes opérations.

Ce cas ne se produit cependant que toutes les 15 interruptions, pour les 14 interruptions restantes, il y a un high du VSYNC et le compteur interne n'est pas affecté.

Mais les signaux HSYNC et VSYNC sont bien sûr nécessaires, de même que DISPEN pour produire le signal vidéo. Une liaison de ces signaux donne le signal SYNC* sur le pin 5 (pin 11) du GA.

1.3.2 La structure des registres du gate array

L'exécution de toutes les tâches que nous avons décrites nécessite que les données soient stockées dans le GA. Le nombre exact des registres internes n'est pas connu mais nous pensons pouvoir décrire les registres les plus importants.

Comme tous les autres éléments du CPC, le GA est appelé à travers l'adressage de port.
Il occupe l'adresse &7Fxx. Il en résulte donc que le bit d'adresse A15 doit être low et le bit d'adresse A14 high. Les autres bits d'adresse (A12 à A8) doivent être mis (sur le niveau high) puisque les autres éléments périphériques sont décodés d'une manière semblablement incompléte. Sur ces périphériques, les entrées de sélection ne sont également reliées qu'aux différents bits d'adresse.
L'état de l'octet d'adresse inférieur est sans importance pour le décodage et n'importe quelle valeur peut y figurer.

On peut distinguer en tout trois différents registres.
Les deux premiers registres sont liés à la production des couleurs, plus précisément aux affectations de couleur fixées avec PEN et INK.
Le premier registre reçoit l'adresse dans laquelle la valeur de couleur doit être écrite. Nous le désignerons désormais sous le nom de registre du numéro de couleur (reg NC).
La valeur de la couleur elle-même peut être ensuite écrite dans le second registre (sous la même adresse de port!). Nous appellerons ce registre registre de valeur de couleur (reg VC).

Le troisième registre est un registre multi-fonctions (reg MF) qui détermine le mode d'écran et la configuration de la mémoire. La sélection des différentes possibilités y est déterminée par les différents bits à l'intérieur du registre.

Dans tous les registres du GA, il n'est possible que d'écrire. Il est IMPOSSIBLE de lire les valeurs de ces registres.

Comme le GA ne peut être appelé qu'à travers une seule adresse de port, il faut qu'il y ait un moyen de distinguer les différents groupes. Cette distinction est opérée grâce aux deux bits supérieurs de l'octet de donnée. Les combinaisons possibles sont:

Bit 7	Bit 6	
0	0	Ecrire une valeur dans le reg NC
0	1	Ecrire une valeur de couleur dans le reg VC choisi
1	0	Ecrire une valeur dans le reg MF
1	1	Utilisé sur le 6128 pour la commutation de mémoires

Mais que représentent les registres de numéro de couleur et de valeur de couleur?
Fondamentalement, ces registres correspondent aux instructions PEN et INK. L'instruction PEN modifie la couleur d'écriture actuelle sur le moniteur. L'affectation d'un numéro PEN à une couleur peut être fixée avec l'instruction INK. Il faut pour cela indiquer le numéro à modifier et la valeur souhaitée. Ce sont exactement ces fonctions qu'exécutent ces deux registres. Le numéro de la couleur à modifier est placé dans le registre NC, aprés quoi la valeur de couleur souhaitée est écrite dans le GA.
Pour modifier par exemple la couleur affectée à PEN 1, il faut employer les instructions suivantes:

```
OUT &7F00,&X00000001:OUT &7F00,&X010XXXXX
```

Dans la première instruction OUT, les bits 6 et 7 valent 0 et les bits 0 à 3 contiennent le numéro de la couleur à modifier. Dans notre exemple, il s'agit du numéro 1. Le bit 5 n'a pas de fonction, le bit 4 a une fonction spéciale sur laquelle nous reviendrons bientôt.
Dans la seconde instruction OUT, les bits 6 et 7 ont été choisis de façon à ce que le registre VC soit sélectionné. Les bits 'X' correspondent simplement à la valeur de couleur. 5 bits permettent en principe de sélectionner 32 couleurs différentes mais il n'y a que 27 couleurs différentes possibles. Les 5 valeurs de couleur restantes sont identiques à d'autres couleurs.

Si vous essayez cet exemple en Basic, vous constaterez que le succès escompté se fait attendre. Tout ce que vous obtenez, c'est un rapide flash de la nouvelle couleur.
La cause en est une particularité du logiciel du CPC. Toutes les couleurs sont représentées en "clignotement". Vous ne le remarquez pas parce que le clignotement ne se fait pas entre couleurs différentes, mais entre couleurs identiques. Lors de chaque commutation entre deux couleurs, tous les paramètres pour le GA sont chargés à nouveau. Mais si, avant les instructions OUT, vous utilisez l'instruction SPEED INK 255,255, vous pourrez observer nettement plus longtemps au moins lors de quelques tentatives l'effet de ces instructions.

Venons-en maintenant à l'explication du bit 4 du reg NC que nous avions différée jusqu'ici. Si ce bit est lors de l'accès fixé sur le registre, l'information des bits 0 à 3 sera ignorée et la valeur de couleur transmise par la prochaine instruction OUT sera interprétée comme nouvelle couleur du bord.

Le registre MF est adressé lorsque, dans l'instruction OUT, le bit 7 est mis et le bit 6 est low. Les autres bits de ce registre ont la signification suivante:

Bit 5: Aucune fonction?
Bit 4: 1 = annuler le compteur V Sync
Bit 3: 1 = déconnecter ROM &C000 à &FFFF
Bit 2: 1 = déconnecter ROM &0000 à &3FFF
Bit 1: Mode écran
Bit 0: Mode écran

Nous n'avons rien pu découvrir jusqu'ici sur la fonction du bit 5.

Si le bit 4 est mis, la chaîne de division pour l'impulsion d'interruption est annulée et le processus de comptage des impulsions V Sync recommence du début. Il serait ainsi possible d'allonger l'intervalle entre deux impulsions d'interruption. Vous pouvez constater cette fonction en Basic grâce à la boucle de programme suivante:

```
10 OUT &7F00,&X10010110:GOTO 10
```

Après avoir lancé cette ligne de programme, vous constatez que l'ordinateur est complètement bloqué et qu'un RESET avec SHIFT/CTRL/ESC n'est même plus possible. Cette ligne provoque en effet une annulation si rapide du registre de comptage, que plus aucune impulsion d'interruption ne peut se produire. Et comme le clavier est interrogé par la routine d'interruption, vous ne pouvez plus réutiliser votre CPC qu'après l'avoir éteint puis rallumé.

Les bits 2 et 3 déterminent la configuration de la mémoire actuelle. Si l'un des bits est mis, c'est la Ram que le processeur rencontrera dans la zone d'adresse correspondante, lors de ses accès en lecture, si ces bits sont nuls, le processeur lira des données dans la Rom.
Une manipulation désordonnée de ces bits débouche au minimum sur des messages d'erreur mais le "plantage" complet du système ou un Reset sont également possibles.

Les bits restants, 0 et 1, déterminent le mode actuel de l'écran. Les combinaisons possibles sont:

Bit 1	Bit 0	
0	0	Mode 0, 20 colonnes, 16 couleurs
0	1	Mode 1, 40 colonnes, 4 couleurs
1	0	Mode 2, 80 colonnes, 2 couleurs
1	1	Comme Mode 0, mais sans clignotement

Si vous avez essayé notre programme d'une ligne pour supprimer les interruptions en mode 1, vous aurez certainement constaté une très curieuse modification des caractères sur l'écran. Dans cet exemple, nous avons choisi comme mode écran le mode 80 colonnes et changé de mode sans vider l'écran. Les caractères représentés se présentent comme s'il manquait des points au milieu de chaque caractère. Vous trouverez l'explication de ce phénomène à la fin du chapitre suivant, lorsque nous décrirons la structure de l'écran et la représentation des caractères.

1.4 Le contrôleur vidéo HD 6845

Le travail principal dans la production de l'image sur le moniteur est accompli par le contrôleur vidéo HD 6845 également désigné comme Cathode Ray Tube Controller, CRTC en abrégé. Ce circuit intégré a été spécialement conçu comme une interface entre des microprocesseurs et des écrans à grille tels que les moniteurs courants.
Il produit, à partir d'un signal d'horloge unique, tous les signaux de synchronisation nécessaires pour le moniteur et tous les paramètres nécessaires à cet effet peuvent être programmés à l'intérieur de limites assez larges.

Avant de décrire l'affectation des pôles de connexion et la structure interne de registres, nous voulons vous donner un bref aperçu des possibilités de cet élément intèressant:

Nombre de caractères par ligne programmable
Nombre de lignes par écran programmable
Matrice de points verticale des caractères programmable
Accès à une zone de mémoire de 16 K
Refresh automatique pour l'utilisation de Rams dynamiques
Fonctions de contrôle du curseur
Curseur programmable (hauteur et clignotement)
Entrée light-pen
Alimentation en 5 volt continu
Entrées/Sorties compatibles TTL

Le 6845 fut développé à l'origine par Motorola pour être employé dans des systémes informatiques dotés de processeurs de la famille 68xx. Mais du fait de son extraordinaire flexibilité et de sa manipulation aisée ce contrôleur se rencontre sur de très nombreux systèmes, y compris sur des systèmes aussi puissants que par exemple Sirius.

1.4.1 Les pôles de connexion du CRTC

La signification des 40 pattes de connexion est la suivante:

MA0-13	:	Memory Adress Lines; les cases mémoire de la mémoire écran sont adressées à travers ces 14 connexions
RA0-4	:	Raster Adress Lines; ces 5 connexions choisissent à partir du générateur de caractères la ligne actuelle de la grille du caractère à représenter
D0-7	:	Bidirectional Data Bus; des informations peuvent être écrites dans le contrôleur et lues à partir de celui-ci à travers ces pins
R/W*	:	Read/Write*; ce signal détermine le sens des données sur les canaux de données. Avec un low, les données peuvent être écrites du processeur dans le CRTC, avec un high elles sont lues à partir du CRTC.
CS*	:	Chip select*. Pour permettre des transferts de données avec le 6845, celui-ci doit être adressé, ce qui est obtenu par un low sur l'entrée CS*.
RS	:	Register Select. Ce signal est utilisé pour choisir entre le registre d'adresse et 18 registres de contrôle. Avec un niveau low sur RS, on peut accéder au registre d'adresse, avec un high, on accède au registre de contrôle.
EN	:	Enable. Avec une bascule ascendante de ce signal, les signaux du processeur se trouvant sur le circuit intégré sont pris en compte.
RES*	:	Reset*.Adress Lines; les cases mémoire de la mémoire écran sont adressées à travers

Vss	1	VSYNC
RES*		HSYNC
LPSTB		RA 0
MA 0		RA 1
MA 1		RA 2
MA 2		RA 3
MA 3		RA 4
MA 4		D 0
MA 5		D 1
MA 6		D 2
MA 7		D 3
MA 8		D 4
MA 9		D 5
MA 10		D 6
MA 11		D 7
MA 12		CS*
MA 13		RS
DISPTMG		E
CUDISP		R/W*
Vcc		CCLK

1.4.1.1 PINOUT du CRTC HD 6845

ces 14 connexions

CLK : Character Clock est le signal d'horloge dont sont tirés par division tous les signaux dont a besoin le moniteur.

HSYNC : Horizontal Sync fournit le signal de synchronisation horizontale du moniteur. La mauvaise définition ou l'absence de HSYNC se traduit par un défilement de l'image.

VSYNC : Vertical Sync fournit le signal de synchronisation verticale du moniteur.

DISPTMG : Display Timing. Ce signal est high lorsque le signal envoyé au moniteur doit être représenté à l'écran. Ce signal permet d'inhiber les retours en arrière du faisceau

CUREN : Cursor Enable (souvent appelé également Cursor Display ou CURDISP) est utilisé lorsque le curseur n'est pas commandé par logiciel mais par le CRTC lui-même. Cette connexion permet également de commander le clignotement du curseur.

LPSTB : Light Pen Strobe. Si une bascule low-high est envoyée sur cette entrée, l'état actuel des canaux MA est transféré et stocké dans les registres Light-pen. Ces registres peuvent être lus pour être utilisés dans un programme.

1.4.2 Les registres internes du contrôleur vidéo

Comme nous l'avons déjà indiqué, le 6845 contient un registre d'adresse et 18 registres de contrôle. Comme le signal RS, Register Select, ne permet toutefois de choisir qu'entre deux adresses, on peut donc se demander comment il est possible d'appeler les 18 registres de contrôle à travers une seule adresse. La solution de ce problème réside dans le registre d'adresse. Le numéro du prochain registre de contrôle auquel on veut accéder est écrit dans le registre d'adresse. Ce procédé semble certes relativement compliqué mais il présente un avantage indéniable. De cette façon en effet, le CRTC n'occupe justement que deux adresses et non pas 18 ou même 32.

Comme d'autre part le CRTC n'est normalement programmé qu'une seule fois, lors de la mise sous tension de la machine, ce travail de programmation supplémentaire est tout à fait acceptable.

Mais examinons maintenant les 18 registres un peu plus en détail. La description suivante semblera peut-être un peu sèche et difficilement compréhensible à cause de la structure complexe des différents registres. De même, certaines connaissances de base en technique vidéo sont nécessaires pour la compréhension de certains registres. Si vous ne comprenez pas tout à la lecture de cette description, consolez-vous en vous disant que le contrôleur vidéo de votre ordinateur n'a pas absolument à être programmé "manuellement".

Dans la présentation suivante, un R placé à la suite du nom du registre indique que ce registre doit être lu et un W signifie qu'on peut écrire dans ce registre. Remarquez que certains registres peuvent uniquement être lus ou écrits, ce qui est indiqué par -.

AR -/W : Adress Register. Ce registre 5 bits reçoit le numéro du registre de contrôle souhaité. Les valeurs de registre 18 à 31 sont ignorées, les seules valeurs valables vont de 0 à 17. Ce registre est appelé lorsqu'aussi bien CS que RS sont low.

R0 -/W : Horizontal Total. Ce registre 8 bits reçoit le nombre de caractères par ligne complète. Notez d'ailleurs qu'une ligne complète est nettement plus grande que les caractères visibles à l'écran car les durées pour le bord et le retour en arrière du faisceau doivent être également prises en compte dans le calcul. Cette valeur est donc environ 1 fois et demi plus importante que le nombre de caractères par ligne choisi.

R1 -/W : Horizontal Displayed. Ce registre contient le nombre de caractères à représenter à l'écran. La valeur placée ici doit être inférieure à celle de R0.

R2 -/W : Adress Register. Ce registre 5 bits reçoit le numéro du registre de contrôle souhaité. Les valeurs de registre 18 à 31 sont ignorées, les seules valeurs valables vont de 0 à 17. Ce registre est appelé lorsqu'aussi bien CS que RS sont low.

R3 -/W : Sync Width. Les 4 bits inférieurs de ce registre déterminent la largeur des impulsions HSync et VSync. Les 4 bits supérieurs de ce registre ne sont pas utilisés.

R4 -/W : Vertical Total. Les 7 bits inférieurs de ce registre déterminent le nombre total de lignes de grille par image. Cette valeur détermine donc ainsi également si la fréquence de renouvellement de l'image est de 50 ou 60 Hertz.

R5 -/W : Vertical Total Adjust. Les 6 bits inférieurs de ce registre permettent de réaliser un ajustement précis de la fréquence de renouvellement de l'image.

R6 -/W : Vertical displayed. Les 7 bits inférieurs de ce registre déterminent le nombre de lignes de grille réellement représentées sur le moniteur. Théoriquement, on peut programmer ici n'importe quelle valeur inférieure au contenu de R4.

R7 -/W : Vertical Sync Position. La valeur 7 bits de ce registre détermine le moment de l'impulsion VSync. Si la valeur de R7 est diminuée, l'image du moniteur est alors décalée vers le bas, si cette valeur est augmentée, il y a décalage vers le haut.

R8 -/W : Interlace. Les deux bits inférieurs de ce registre permettent de déterminer si la représentation doit avoir lieu avec ou sans procédure de saut de ligne (interlace).

R9 -/W : Maximum Raster Adress. Ce registre 5 bits détermine le nombre de lignes de grille des caractères à représenter.

R10 -/W : Cursor Start Raster. Les bits 0 à 4 de ce registre déterminent sur quelle ligne de la grille doit commencer le curseur. Les bits 5 et 6 fixent le mode de curseur de la façon suivante:

Bits 6	5	
0	0	Curseur non clignotant
0	1	Curseur non représenté
1	0	Curseur clignotant (env. 3 par seconde)
1	1	Curseur clignotant (env. 1.5 par seconde)

R11 -/W : Cursor End Raster. En fonction du contenu de R10, les 5 bits inférieurs de ce registre déterminent sur quelle ligne de l'écran se termine le curseur.

R12 R/W : Start Adress High. Les bits 0 à 5 déterminent à partir de quelle adresse de tout le domaine d'adressage de 16 K du CRTC commence la mémoire écran. Si ce registre est lu, les bits 6 et 7 sont toujours low.

R13 R/W : Start Adress Low. Ce registre fixe, de façon analogue à R12 l'octet d'adresse faible de la mémoire écran à adresser.

R14 R/W : Cursor High. Les bits 0 à 5 de ce registre représentent l'octet fort de la position actuelle du curseur.

R15 R/W : Cursor Low. De façon analogue à R14, ce registre reçoit l'octet faible de l'adresse du curseur.
Comme R14 ainsi que R15 peuvent être écrits ou lus, ces registres permettent de déterminer librement la position du curseur.

R16 R/-: Ce registre contient après une impulsion strobe positive l'octet fort de l'adresse de la mémoire écran qui était activée au moment de l'impulsion. Les bits 6 et 7 de ce registre sont toujours low.

R17 R/-: De façon analogue à R16, ce registre contient l'octet faible au moment du strobe light-pen.
R16 ainsi que R17 ne peuvent qu'être lus.

1.5 La Ram du CPC

La RAM (mémoire écriture/lecture) de 64 K intégrée dans le CPC n'est pas uniquement utilisée comme mémoire de donnée et de programme. Les informations concernant l'écran sont également placées dans cette mémoire.

Après que nous ayons étudié en détail dans les chapitres précédents les trois éléments les plus importants, le processeur, le gate array et le contrôleur vidéo, nous allons dans le présent chapitre jeter un regard sur l'interaction de ces trois éléments lors de l'accès aux circuits intégrés de mémoire. Nous expliquerons également à cette occasion comment le contrôleur vidéo appelle la Ram pour représenter des caractères à l'écran.

Mais nous voulons faire auparavant une petite digression pour étudier comment fonctionnent les éléments de mémoire.

Nous allons tout d'abord expliquer comment est possible l'adressage de 65536 cases mémoire avec les 8 connexions d'adresse existantes.
Le principe consiste à diviser l'adresse 16 bits en deux moitiés et à envoyer ces deux octets d'adresse l'un après l'autre sur les pins d'adresse de la Ram. Ce procédé est appelé multiplexage. Le multiplexage nécessite cependant des signaux qui indiquent à la Ram quelle information se trouve dans l'instant sur les connexions d'adresse.

C'est ici qu'entrent en jeu les signaux RAS* et CAS* fournis par le gate array.
Après qu'un octet d'adresse ait été envoyé aux Rams, une bascule high-low du signal RAS* leur indique qu'une moitié d'adresse est prête. Avec la bascule négative (high-low) du RAS*, l'information d'adresse disponible est stockée dans les Rams.
La deuxième moitié de l'adresse peut alors être envoyée à la Ram. Dès que cet octet d'adresse est prêt, le signal CAS* devient low. La Ram a ainsi reçu la totalité de l'adresse 16 bits et sélectionne alors la case mémoire souhaitée. Il est maintenant possible d'écrire ou de lire cette case.

La commutation des moitiés d'adresse doit bien sûr être également prise en charge par un signal convenable, sur le CPC, c'est le signal CAS-ADDR*.
Le multiplexage ou commutation est effectué par quatre circuits intégrés du type 74LS153. On peut se représenter le fonctionnement de ces circuits intégrés du type 74LS153 comme deux commutateurs commandés électroniquement. A travers deux entrées de commande, on peut décider laquelle des quatre entrées doit être reliée à la sortie.
Les deux entrées de commande sont commandées par les signaux CPU-ADDR* et CAS-ADDR*. Le signal CPU-ADDR* permet de décider si c'est le processeur ou le CRTC qui peut envoyer une adresse à la Ram et CAS-ADDR* effectue la commutation entre les moitiés d'adresse.

La table suivante montre l'affectation précise des connexions d'adresse du processeur et du contrôleur vidéo:

Z80	6845	Z80	6845
A0	CCLK	A8	MA7
A1	MA0	A9	MA8
A2	MA1	A10	MA9
A3	MA2	A11	RA0
A4	MA3	A12	RA1
A5	MA4	A13	RA2
A6	MA5	NA14	MA12
A7	MA6	NA15	MA13

Comme on voit, tous les bits d'adresse du processeur sont envoyés à travers les multiplexeurs sur les connexions d'adresse des Rams. Sur le CPC 6128, toutefois, les signaux d'adresse A14 et A15 ne sont pas placés directement sur les multiplexeurs. C'est en effet ici qu'est intercalé le composant responsable de la commutation de la mémoire. Mais le contrôleur vidéo adresse également avec l'aide du CCLK l'ensemble de la zone adressable de 64 K. Ce qui contredit cependant le chapitre précèdent où nous disions que le CRTC ne peut adresser qu'une zone de 16 K.

Cette affirmation était exacte dans la mesure où seules les 14 connexions désignées par MA (Memory Adress Line) peuvent être comptées comme canaux d'adresse. Ces 14 connexions permettent d'adresser une zone d'adresse de 16 K.

Le mode de travail du 6845 utilisé dans le CPC pour l'adressage de la mémoire vidéo est rarement employé. Les connexions RA0 à RA4 servent normalement à appeler une Rom de caractères déjà programmée qui contient le modèle bits des caractères qui doivent être représentés à l'écran.
Les ordinateurs ont normalement une zone de mémoire appelée mémoire vidéo dans laquelle sont stockés tous les caractères à représenter à l'écran. Dans cette mémoire, l'emplacement de chaque caractère occupe un octet. Cela donne donc, pour représenter 80 x 25 caractères, une mémoire de 2000 octets.
Mais il n'est pas possible de faire entrer dans un octet toutes les informations nécessaires pour la représentation des caractères. Chaque caractère se compose en effet d'un certain nombre de lignes de points placées les unes sous les autres.

Sur le CPC, on peut également reconnaître ces lignes sur le moniteur. C'est ainsi par exemple que le curseur se compose de 8 lignes placées les unes sur les autres, dont tous les points image sont "allumés". Pour représenter des lettres ou des chiffres, seuls les points d'une ligne correspondant à la forme de la lettre ou du chiffre sont allumés. Les modèles de ces lignes de points sont stockées sous forme de cartes bits, dans lesquelles un bit mis correspond normalement à un point allumé sur l'écran.
Les connexions RA sont maintenant nécessaires pour recevoir de la Rom de caractères les différentes lignes, c'est-à-dire les cartes bits. A cet effet, les connexions RA sont utilisées comme canaux d'adresse pour la Rom de caractères.

Comme vous pouvez l'imaginer, il n'est pas possible de réaliser à l'écran du graphisme haute résolution lorsqu'on utilise une Rom de caractères. Les ordinateurs fonctionnant suivant ce principe ne peuvent sortir du jeu de caractères intégré.
Sur le CPC, cette Rom de caractères n'existe pas et on a choisi une voie totalement différente.

Comme les connexions RA adressent directement la mémoire, les informations sur les points doivent donc nécessairement figurer également en Ram. Ce n'est qu'à travers cette astuce de commutation qu'il est possible de produire n'importe quelle carte bits sur le moniteur et donc de représenter le graphisme dans les limites connues.

Mais avant que nous ne nous tournions vers la structure concrète de la mémoire vidéo, il nous faut enfin expliquer le signal CCLK. Mais il nous faudra pour cela un peu de mathématiques.

Le CRTC est commandé par une fréquence d'horloge de 1 MHz. Avec chaque impulsion d'horloge est adressée une case mémoire. Dans cette case se trouvent les informations sur les points qui doivent être représentés 'allumés' sur l'écran, c'est-à-dire dans la couleur d'écriture. Comme une fréquence de 1 MHz correspond à une période de 1 micro-seconde, exactement un huitième de la fréquence d'horloge est disponible pour la réprésentation de chaque point, soit 0.125 micro-secondes. Pour représenter les 640 points d'une ligne, il faut donc un temps de 80 micro-secondes.

Mais comme le signal V Sync qui détermine la durée d'une ligne a une période de 52 micro-secondes, le compte n'est pas bon. Ces valeurs ne permettent de réprésenter que 40 caractères au maximum.

La solution à ce problème réside dans un mode spécial de travail des Rams, le Page Adress-Mode (mode d'adressage par page). Lorsqu'une Ram, après avoir envoyé les signaux RAS et CAS, envoie le contenu de la case mémoire souhaitée sur les sorties de donnée, il suffit alors de n'envoyer avec une autre impulsion CAS qu'une nouvelle moitié d'adresse aux Rams pour recevoir l'octet suivant. Cela suppose bien sûr que seule une moitié des informations d'adresse change.

C'est exactement cette possibilité qu'ont utilisée les développeurs du CPC. Bien sûr, il faut que les informations d'adresse correspondant aux deux différentes impulsions CAS soient différentes, sinon on lit deux fois la même case mémoire. Mais c'est justement ce que réalise le signal CCLK qui commute exactement entre les deux impulsions CAS. Ce signal est envoyé par le multiplexeur IC 105 sur le bit d'adresse 0 (du point de vue du processeur), lorsque le signal CAS-ADDR est sur low et le signal CPU-ADDR par contre sur high. Ce signal représente bien ainsi le bit d'adresse inférieur de la Ram vidéo.

Les deux octets fournis rapidement l'un après l'autre par la Ram vidéo sont entrestockés dans le gate array, convertis dans la forme sérielle indispensable pour le moniteur et envoyés avec les informations de couleur sur la sortie RVB.

Restent encore les deux signaux MA12 et MA13. Ces deux signaux permettent de déterminer par blocs de 16 K le début de la Ram vidéo. Ces bits sont normalement mis et la Ram vidéo commence donc en &C000. Mais il est également possible d'obtenir par programmation que la Ram vidéo soit placée de &4000 à &7FFF.

1.5.1 Les 64 K supplémentaires du 6128

L'analyse du fonctionnement de la commutation des mémoires dans le 6128 n'a pas été sans nous poser quelques problèmes. Il n'était pas possible de résoudre ce problème par de simples PEEKs et POKEs. La seule chose à laquelle nous pouvions nous raccrocher était le programme 'BANKMAN' livréavecl'ordinateur.Commeceprogramme, pour des raisons incompréhensibles, est toutefois un programme protégé, les choses ne furent pas très simples. Quoi qu'il en soit, après un certain temps de tests et d'expérimentation, nous sommes en mesure de décrire au moins les principes essentiels de la commutation de mémoires.

Avant que nous ne décrivions cependant la commutation de mémoires, deux notions doivent être expliquées. Par banque, nous entendons une zone mémoire de 64 K octets, alors qu'un bloc a, lui, une taille de 16 K octets. Ces deux termes seront fréquemment employés dans la section suivante.

L'organisation de la mémoire est prise en charge par un composant PAL du type HAL16L8. Sur ce composant sont placés les canaux de données D0 à D2 ainsi que D6 et D7, les signaux d'adresse A14 et A15, le signal IOWR*, le signal CAS ainsi que RESET et CPU*. Les signaux disponibles en sortie sont NA14, NA15, CAS0 et CAS1. Le PAL lui-même occupe l'adresse de port &H7Fxx, exactement comme le gate array. Nous avons indiqué, lors de la description du gate array, que la sélection des registres dans le GA est effectuée par l'état des bits de données D6 et D7. La combinaison pour laquelle les deux bits de données valent 1 (high) ne sélectionne aucun registre du GA. Au lieu de cela, l'information contenue dans les bits de données D0 à D2 est évaluée par le PAL. C'est à travers cette information qu'est effectuée la commutation entre les diverses configurations de la mémoire possibles.

Après une impulsion RESET, l'ordinateur se comporte comme s'il n'y avait qu'une banque de 64 K octets intégrée. Les signaux d'adresse A14 et A15 du Z80 sont transmis sans modification au multiplexeur d'adresse, à travers le PAL.

Le signal CAS du GA est placé, à travers le PAL, sur le pin CAS0. Le signal CAS1 est provisoirement inactif. Ainsi est activée la première banque du CPC lors des accès à la mémoire. Le refresh est par ailleurs également assuré pour la deuxième banque puisque seul le signal RAS est nécessaire à cet effet. Ce signal est cependant placé parallèlement sur les deux banques.

Si l'on sort cependant sur le PAL une valeur appropriée, la situation de la mémoire du CPC change notablement. Mais demandons-nous d'abord quelles valeurs seraient possibles. L'adresse de port est déjà connue. Nous savons d'autre part que les bits de données D6 et D7 doivent être mis pour que nous n'appelions pas involontairement des registres du GA. Les bits de données D3 à D5 ne sont pas interrogés car ils ne sont pas reliés au PAL. Nous savons donc maintenant quelles sont les valeurs possibles: &C0 à &C7. Mais quel est l'effet des différentes valeurs?

Malheureusement, du fait de la structure du CPC, il est très difficile d'analyser toutes les combinaisons. Dans certains cas, c'est en effet pratiquement la totalité de la mémoire qui est commutée. Et bien sûr, après la commutation, le programme de commutation disparaît. Cela aboutit à un "plantage" classique du système. Nous pouvons cependant vous indiquer quelles sont les combinaisons intéressantes pour vous. Avec ces valeurs, la mémoire de la zone d'adresse de &4000 à &7FFF de la banque 0 est échangée contre un bloc de la banque 1. Les valeurs nécessaires à cet effet figurent dans la table suivante:

&C0 Banque 0, bloc 1 (situation de départ)
&C4 Banque 1, bloc 0
&C5 Banque 1, bloc 1
&C6 Banque 1, bloc 2
&C7 Banque 1, bloc 3

Si une des valeurs entre &C4 et &C7 est sortie sur l'adresse de port &7Fxx, le CAS0 devient inactif dans la zone de &4000 à &7FFF. Le signal CAS1 devient par contre actif. L'information sur les pins d'adresse A14 et A15 du Z80 est modifiée par le PAL.

La valeur &C5 constitue une exception à cet égard puisqu'elle appelle la même zone d'adresse de la deuxième banque. &C4 adressera cependant la zone d'adresse &0000 à &3FFF de la seconde banque, sans que le processeur 's'en rende compte'. Pour lui, la RAM se trouve toujours dans la zone d'adresse qu'il souhaite. Il en va de même pour la valeur &C6 et la zone d'adresse de &8000 à &BFFF, pour &C7 et la zone de &C000 à &FFFF de la seconde banque.

Nous n'avons malheureusement pas pu élucider la signification précise des valeurs &C1 à &C3. Ces valeurs jouent certainement un rôle important sous CP/M 3.0, puisqu'avec les autres valeurs il n'est pas possible de réaliser une TPA de 61 K octets. Toutefois, pour les lecteurs intéressés, nous pouvons vous indiquer quelle est l'organisation de la mémoire sous CP/M 3.0.
Sous ce système d'exploitation, on doit "errer" avec trois zones de RAM parallèles. Bien entendu, vous n'avez pas à régler vous-même la commutation des mémoires, CP/M s'en charge pour vous.

Pour les trois banques, la zone d'adresse de &C000 à &FFFF est identique. Cette zone n'est jamais commutée car c'est à travers elle qu'est entreprise la commutation des autres zones. Dans cette zone figure la limite supérieure de la TPA, ainsi que les sections du BIOS et du BDOS qui sont toujours résidentes.

Dans la banque 0 figurent encore trois autres blocs. Le premier bloc (&0000 à &3FFFF) contient le bloc jump inférieur qui est copié, dès la mise sous tension, de la ROM dans le bas de la RAM. Dans le bloc 1 de la banque 0 (&4000 à &7FFF) se trouve la RAM écran. Dans le bloc 2, enfin, se trouve la plus grande partie du BIOS et du BDOS ainsi que les blocs jump nécessaires qui empêchaient en effet, sous CP/M 2.2, une extension de la TPA.
La banque 2 se compose des blocs 0 à 2 et contient la plus grande partie de la TPA. La partie encore manquante de la TPA figure dans le bloc 3 de cette banque. Ce bloc est cependant identique pour les trois banques.
La banque 2 contient enfin encore une fois la zone de &4000 à &7FFF. Cette zone contient le CCP et les tables hash nécessitées par CP/M.

Si vous voulez expérimenter les valeurs de commutation de mémoires qui n'ont pas été expliquées dans ce chapitre, nous vous invitons à suivre les conseils suivants. Vous devriez d'abord transférer la mémoire écran de &C000 à &4000. Vous devriez ensuite placer, avant de le lancer, votre programme de test dans la zone en &C000 ainsi libérée. L'idéal serait bien sûr un petit programme de moniteur placé dans cette zone. La raison en est que cette zone n'est sans doute jamais déconnectée, contrairement à ce qui peut arriver pour les autres zones. Le programme de moniteur doit cependant restaurer la configuration originelle de la mémoire avant tout appel de routines système à travers les blocs jumps. Il faut donc sortir la valeur &C0 sur l'adresse de port appropriée. Si vous oubliez cette restauration, il peut arriver que, du fait de la commutation que vous avez effectuée, les blocs jumps ne soient absolument pas accessibles. Dans ce cas, l'ordinateur se "plante" en beauté.

1.6 La Ram vidéo entre Z80 et 6845

Essayez maintenant ce petit programme sur le CPC:

```
10 MODE 2
20 FOR i=&c000 TO &ffff
30 POKE i,255
40 NEXT i
```

Vous obtenez sur l'écran une ligne étroite qui est rapidement dessinée vers la droite à partir de l'angle supérieur gauche de l'écran. A la fin de la première ligne, le dessin se poursuit exactement 8 lignes plus bas.
Une fois dessinées ces lignes étroites sur toute la surface de l'écran, le dessin reprend d'en haut mais en dessous des lignes déjà dessinées.

Essayez le programme également en mode 1 et en mode 0.

Puis modifiez aussi la ligne 30 ainsi:

```
30 POKE i,1
```

Vous obtenez maintenant une ligne de points qui remplit l'écran verticalement.

Lorsque le programme tourne en mode 2, on voit que les lignes verticales se trouvent sur le côté droit des caractères.
En mode 1, nous obtenons 2 lignes verticales par caractère, en mode 0, 4 lignes.

Nous allons maintenant apporter une dernière modification au programme. Supprimez la ligne 10 du programme et entrez 'MODE 2' en mode direct. L'écran se vide et Ready apparaît dans l'angle supérieur gauche. Appuyez sur la touche de curseur BAS, jusqu'à ce que le message Ready disparaisse de l'écran. Le curseur se trouve maintenant dans la dernière ligne de l'écran. Faites à nouveau tourner le programme.

Le résultat est quelque peu agaçant.

Ce petit programme nous a révélé plusieurs choses importantes d'un seul coup. D'abord nous avons démontré que la mémoire écran commence en &C000 et finit en &FFFF. Curieusement, la taille de la mémoire écran est la même pour les trois modes écran. La seule différence entre les modes réside dans les couleurs.

Cependant on peut se demander à quoi servent 16 K de mémoire écran en mode 0, lorsqu'on ne représente que 20 caractères par ligne. 20 caractères par 25 lignes font 500 caractères sur l'écran. Pourquoi le CPC a-t-il besoin de 16384 cases mémoire pour représenter à l'écran ces 500 caractères?

La réponse est simple. Comme nous l'avons déjà indiqué, le CPC ne possède pas de Ram vidéo dans laquelle chaque caractère serait stocké dans un octet.
En mode 80 colonnes, un caractère sur l'écran occupe 8 octets, en 40 colonnes, un caractère occupe 16 octets et 32 octets en mode 20 colonnes. C'est d'ailleurs ce que montrait le programme qui produisait les lignes verticales.

Le mode 80 colonnes est à cet égard le plus simple à comprendre, puisque chaque bit mis produit un point dans la couleur actuelle d'écriture (pen). Si un bit n'est pas mis, c'est au contraire la couleur du fond de l'écran qui apparaît à cet endroit. Comme en mode 2, il n'y a qu'une couleur d'écriture possible, il n'y a pas d'autres possibilités.

Mais à quoi servent donc en mode 0 les 32 octets nécessaires pour un caractére?

Le fonctionnement des modes 0 et 1 n'est plus aussi simple à expliquer. Nous vous conseillons de taper le petit programme suivant et d'avoir sous les yeux les effets de ce programme, pendant que vous lirez nos explications. Les explications seront alors plus compréhensibles.

```
10 MODE 2
20 REM
30 PRINT "A"
40 FOR adresse=&c000 TO &f800 STEP &800
50 p$=BIN$(PEEK(adresse),8)
60 FOR I=1 TO 8
70 IF MID$(p$,I,1)="1" THEN PRINT "X"; ELSE PRINT ".";
80 NEXT I
90 PRINT
100 NEXT adresse
```

Faites tourner ce programme et vous obtiendrez une image correspondant à la matrice de 'A'.

Modifiez maintenant l'instruction MODE de la ligne 10 en MODE 1 et faites tourner le programme. Le résultat est assez surprenant.
Vous pouviez vous imaginer que seule une moitié de la matrice figurerait dans les octets lus. Mais il semble curieux a priori que la matrice n'utilise qu'une moitié d'octet, soient les bits 4 à 7.

Mais nous nous rapprochons de la solution de cet énigme, lorsque vous modifiez ainsi la ligne 20:

```
20 PEN 2
```

Non seulement la couleur d'écriture (PEN) s'est modifiée, mais la carte bits montrée par notre programme s'est aussi modifiée. Et voilà la solution de notre problème!

Si vous connaissez déjà le CPC, vous savez qu'en mode 40 colonnes, 4 couleurs sont possibles. Ces 4 couleurs sont tout simplement stockées avec le caractère lui-même. En effet 4 bits seulement déterminent les pixels (points de l'écran) allumés et les quartets low et high décident des couleurs (un quartet=un demi-octet, 4 bits). Avec le principe utilisé, le gate array n'a qu'à doubler horizontalement les pixels correspondant à l'affichage, représentant ainsi 8 points, alors que seuls 4 bits sont stockés en mémoire.

En mode 0, pour représenter 20 caractères par ligne, cette méthode est encore étendue. Il n'y a plus ici que deux bits qui contiennent les informations sur les pixels. La position de ces deux pixels à l'intérieur de l'octet détermine la couleur dans laquelle ces pixels doivent être représentés. Il y a ainsi 16 combinaisons possibles, ce qui correspond exactement au nombre de couleurs disponibles. Comme seulement deux pixels sont stockés dans un octet, 4 x 8 = 32 octets sont nécessaires pour représenter un caractère avec 16 couleurs différentes possibles.

Essayez à nouveau le programme en mode 0 en utilisant différentes valeurs pour l'instruction PEN. Vous comprendrez vite le fonctionnement.

Les deux premiers points soulevés au début du chapitre sont ainsi éclaircis. Reste cependant le point du 'décalage' de la Ram écran. Ce problème a sa source dans l'électronique du CPC.

Même un Z80 avec une fréquence d'horloge de 4 MHz a besoin d'un certain temps pour décaler un bloc de données de 16 K. Par exemple, pour éviter d'avoir à décaler de 640 cases mémoire, lors du listage d'un programme assez long, la totalité de la zone de Ram vidéo, on a utilisé une propriété du CRTC. Par programmation adéquate des registres 12 et 13 du 6845, l'écran peut commencer pratiquement en n'importe quelle case mémoire paire de la Ram vidéo. Le scrolling (défilement de l'écran) peut ainsi se produire nettement plus vite, puisqu'il suffit de fournir les valeurs adéquates aux registres qui conviennent. La nouvelle ligne dans le bord inférieur de l'écran est vite effacée et remplacée par les nouveaux caractères.

Il n'est pas possible de faire commencer la Ram vidéo à une adresse impaire, par exemple en &C001, du fait de l'utilisation décrite plus haut du signal CCLK comme bit d'adresse.

Le programme suivant montre qu'il est possible de manipuler les registres décrits, même en Basic:

```
10 adrreg = &bc00 : REM registre d'adresse du 6845
20 datreg = &bd00 : REM port du registre de donnee
30 OUT adrreg,13 : REM selectionner le registre
40 FOR offset = 1 TO 40
50 OUT datreg,offset : REM modifier 40 fois
60 for attendre = 1 TO 40 : REM et attendre un peu
70 NEXT attendre,offset
```

Ce programme réalise un scrolling horizontal de l'écran. Sans la boucle de temporisation, le scrolling se déroulerait tellement vite qu'il ne serait pas possible de suivre avec un oeil humain.

Le scrolling vertical peut également être programmé en Basic. Il faut alors modifier les deux registres low-byte et high-byte. Mais comme il s'écoule beaucoup de temps entre les deux instructions OUT, on obtient des phénomènes désagréables à l'écran.

Mais, en ce qui concerne la Ram vidéo, il faut encore relever une particularité.

Multiplions les valeurs que nous connaissons entre elles.
En mode 2, un caractère se compose de 8 octets. Il y a 80 caractères par ligne et 25 lignes sur l'écran. La place occupée en mémoire est donc de 80 x 25 x 8 = 16000 octets. Mais une zone de 16 K comporte 2 puissance 14 = 16384 emplacements. Où sont les 384 octets manquants?
Très simple. Ils ne servent à rien, du moins tant qu'il n'y a pas de scrolling de l'écran.
Il est donc possible de placer ici des valeurs à stocker provisoirement. Ces valeurs seront cependant effacées par la première instruction CLS.

Vous vous demandez certainement comment il est donc possible de programmer du graphisme avec une organisation aussi compliquée de la mémoire écran.
Il semble également impossible de lire un caractère à partir de l'écran. Sur d'autres ordinateurs, cela ne pose pas de problème, puisqu'on peut placer un caractère sur l'écran avec POKE et qu'on

peut donc lire le contenu de la Ram vidéo avec PEEK.
D'autre part il est normalement assuré que la Ram vidéo commence à une adresse déterminée.

Les choses ne se présentent cependant pas aussi mal que cela peut sembler au premier abord. Le système d'exploitation est en effet en mesure de discerner les adresses de début modifiables ou de déterminer un caractère à partir de la matrice de l'écran, comme cela se produit chaque fois que vous utilisez la touche COPY. Les routines utilisées à cet effet peuvent également être employées dans des programmes en langage-machine que vous aurez réalisés vous-même.
Vous retrouverez bon nombre de ces routines du système d'exploitation dans un prochain chapitre. Nous vous montrons concrètement comment utiliser le graphisme dans un exemple de dessin de rectangles et dans un programme de hardcopy graphique.

1.7 L'interface parallèle 8255

Développé à l'origine par INTEL pour le 8080, le 8255 convient également pour d'autres processeurs comme élément polyvalent d'entrée/sortie. Le 8255 dispose en tout de 24 canaux à travers lesquels les signaux peuvent être sortis ou entrés. Chaque groupe de 8 canaux constitue un port 8 bits et le troisième port peut être scindé en deux moitiés programmables.

Les principales caractéristiques du 8255 sont:

24 connexions d'entrée/sortie programmables
Alimentation en courant continu de 5 volts
Entièrement compatible TTL
Trois puissants modes de travail programmables
Chaque port programmable séparément
Courant de sortie de 1 mA pour une tension de 1.5 Volts
Possibilité de fonction mettre bit/annuler bit

1.7.1 L'affectation des connexions du 8255

L'affectation des pins du 8255 est indiquée par la figure suivante. En voici la signification:

D0 - D7 : Data lines. Ces connexions sont reliées au bus de données du processeur. Elles servent au transfert des données vers et à partir du processeur.

CS: Chip select. Un low sur ce pôle sélectionne le composant. Les signaux figurant actuellement sur les canaux RD, WR et Data sont acceptés par le 8255.

RD Read. Un low sur ce pôle entraîne que le 8255 envoie des données ou des informations d'état au processeur, à travers le bus de données.

WR Write devient low lorsque le processeur veut envoyer des données ou des instructions de commande au 8255.

A0,A1 : Adress Lines 0 et 1. A travers ces connexions s'opère la sélection entre les trois canaux de données et le registre de commande. Ces connexions sont souvent liées aux deux canaux d'adresse inférieurs du processeur.

RESET : Un high sur cette entrée rétablit les valeurs initiales de tous les registres, y compris le registre de commande. Les canaux de port sont placés en mode de travail entrée.

PA0 - PA7 : Port A. Ces huit canaux représentent le port d'entrée/sortie A et peuvent être utilisés au choix en entrée ou en sortie.

PB0 - PB7 : Port B. Fonctionnement identique au port A

PC0 - PC7 : Port C. Fonctionnement identique au port A

PA 3 (1)	PA 4
PA 2	PA 5
PA 1	PA 6
PA 0	PA 7
RD*	WR*
CS*	RESET
GND	D 0
A 1	D 1
A 0	D 2
PC 7	D 3
PC 6	D 4
PC 5	D 5
PC 4	D 6
PC 0	D 7
PC 1	Vcc
PC 2	PB 7
PC 3	PB 6
PB 0	PB 5
PB 1	PB 4
PB 2	PB 3

1.7.1.1 PINOUT du PORT PARALLELE 8255

1.7.2 Les modes de travail du 8255

Avant que nous n'en venions aux quatre registre internes, il nous faut tout d'abord examiner d'un peu plus près les possibilités de ce circuit intégré. Comme nous l'avons indiqué au début, le 8255 dispose de 3 modes de travail possibles:

Mode de travail 0 : Simple entrée/sortie
Mode de travail 1 : Entrée/sortie manipulable
Mode de travail 2 : Bus à deux sens

Le mode de travail 0 est le plus simple et le plus courant. Dans ce mode, il est possible de déterminer si les ports doivent travailler comme canaux d'entrée ou de sortie. Si les canaux sont programmés comme sortie et si le processeur envoie une information sur ces sorties, cette valeur est stockée, et les sorties sont conservées jusqu'à une nouvelle programmation ou jusqu'à un reset.
Les ports programmés comme entrée fournissent lors d'une lecture l'état momentané de ces canaux.

Le sens des données sur le port A aussi bien que sur le port B ne peut être programmé que de façon identique pour tout le port. Il n'est pas possible d'utiliser par exemple les bits de port PA0, PA3 et PA7 en sortie et les autres bits du même port en entrée.
Le port C peut cependant être programmé en deux moitiés distinctes. Le sens des données de chaque moitié peut être programmé séparément.

Le mode de travail 1 se différencie fondamentalement du mode 0. Dans ce mode de travail, un transfert de données dans un sens est possible avec des signaux hand shake. On ne parle plus alors de trois ports car les deux moitiés du port C sont mises à la disposition des deux autres ports comme signaux de commande et de réception. On parle alors des deux groupes A et B.

Le groupe A comprend le port A et les bits 4 à 7 du port C, le groupe B le port B et les bits 0 à 3 du port C.
Pour programmer facilement le mode 1, il est possible d'utiliser un bit spécial de chaque moitié du port B comme signal d'interruption.

Un tel transfert de données 8 bits est utilisé par exemple sur les interfaces d'imprimante. Un signal indique ici que les données sur les canaux de données sont valables. Un signal rapporté indique si le récepteur, en l'occurence l'imprimante, est prêt à recevoir des données, ou si les données ont été reçues correctement.
Cette fonction peut être exécutée par le 8255, au choix pour une sortie ou une entrée de données.

Le troisième mode de travail (mode 2) est un mode de travail bidirectionnel. Cette fonction n'est possible qu'avec le port A. Les bits PC3-7 sont utilisés comme signaux de commande et de réception.
Une application possible de ce mode de travail serait la commande d'un lecteur de disquette car les données doivent dans ce cas être transmises aussi bien du lecteur de disquette au processeur que du processeur au lecteur, à travers les mêmes connexions.

Il est d'autre part possible dans les trois modes de travail de mettre ou d'annuler, individuellement, par instruction, les bits programmés en sortie.
Les trois modes de travail ainsi décrits peuvent être également combinés. Il est ainsi possible d'utiliser le Port A en mode 0 comme sortie, le port B en mode 1 comme entrée et de programmer les bits restants du port C en entrée.

1.7.3 Commande du 8255, description des registres

Lorsqu'on considère tout d'abord ce nombre troublant de possibilités, on se demande malgré soi comment toutes les possibilités et combinaisons peuvent être programmées avec un seul registre de commande.
L'astuce qui rend cela possible est simple. Le bit supérieur du mot de commande est utilisé comme bit témoin. Si ce bit est mis dans le mot de commande, les bits 0 à 6 ont la signification suivante:

Bit 0 : commande la fonction Port C bits 0-3
1=Entrée
0=Sortie

Bit 1 :   commande la fonction Port B
1=Entrée
0=Sortie

Bit 2 :   sélectionne le mode groupe B
1=Mode de travail 0
0=Mode de travail 1

Bit 3 :   commande la fonction Port C bits 4-7
1=Entrée
0=Sortie

Bit 4 :   commande la fonction Port A
1=Entrée
0=Sortie

Bit 6,5 :   sélectionne le mode groupe A
00=Mode 0
01=Mode 1
1x=Mode 2, bit 5 sans signification

Si par contre le bit supérieur du mot de commande est nul, la fonction 'mettre un bit/annuler un bit' du port C est définie. La signification de ces bits est:

Bit 0 :   commande Bit Set/Bit Reset
1=mettre un bit
0=annuler un bit

Bits 3-1:   Sélection du bit

000 = PC0
001 = PC1
010 = PC2
011 = PC3
100 = PC4
101 = PC5
110 = PC6
111 = PC7

Les bits 4 à 6 du mot de commande sont sans signification lorsque le bit 7 est nul.

Ce registre de commande ne peut être lu. Il n'est possible que d'y écrire. Les registres correspondant aux ports peuvent par contre être lus, même lorsque les ports sont utilisés en sortie. Dans ce cas, la valeur lue correspond à l'état des canaux de port.

L'accès aux quatre registres se fait à travers les pins de connexion A0 et A1. Ces connexions sont décodées dans le 8255 et utilisées comme signaux de sélection de registre. Normalement A0 et A1 du 8255 sont envoyés sur les canaux de même nom du processeur. Il en résulte un adressage transparent sur 4 adresses.
L'affectation aux registres des connexions A0 et A1 est indiquée par le tableau suivant:

A1	A0	
0	0	Registre Port A
0	1	Registre Port B
1	0	Registre Port C
1	1	Registre de commande

1.7.4 L'utilisation du 8255 sur le CPC

Après avoir donné un aperçu des possibilités variées du 8255, nous en venons au fonctionnement pratique de ce composant universel sur le CPC. Comme en fait presque tous les circuits intégrés sur le CPC, le 8255 est également utilisé de façon optimale. Aucun bit n'est inutilisé.

Mais devenons plus concret.
Le 8255 sert le clavier, le chip sonore, le moteur du lecteur de cassette, produit les signaux d'écriture du lecteur de cassette, lit le flux de bits venant du lecteur de cassette, contrôle le signal V Sync du CRTC, contrôle si l'imprimante est prête à recevoir, interroge avec un bit l'état du signal EXP du connecteur d'extension, décide à travers un pont si la production de l'image doit se faire suivant la norme PAL ou SECAM en 50 ou 60 Hertz et il reste enfin encore trois bits qui interrogent des ponts lors de la

mise sous tension de façon à savoir quel ordinateur vous avez acheté. L'état de ces ponts décide en effet si vous recevrez dans le message d'initialisation, le nom de la firme Amstrad, Awa, Triumpf, Schneider ou un autre des 8 noms possibles.

Avoir réalisé toutes ces fonctions avec uniquement les 24 canaux d'entrée/sortie disponibles, témoigne de l'esprit d'économie et de l'inventivité des développeurs de ce matériel.

Le bus de données est relié directement au bus de données du processeur. Le signal CS (Chip Select) est produit par le bit d'adresse A11 du processeur. Les pins A0 et A1 du 8255 pour la sélection de registre sont reliés aux pins d'adresse A8 et A9 du processeur.
Comme nous l'avons déjà indiqué, les éléments périphériques du CPC sont appelés à travers des adresses de port. C'est pourquoi le canal RD* du 8255 est relié au signal IORD*.
Ce signal est produit par la combinaison des signaux RD* et IORQ* du Z80. Uniquement lorsque IORQ* et RD* sont low, apparaît un low sur l'entrée RD*.
La connexion WR* du 8255 est commandée de même. Ici apparaît un low, venant du pin 3 du 74LS32, lorsqu'aussi bien WR* que IORQ* du Z80 deviennent low.

Ces données permettent maintenant de déterminer les adresses de port du 8255. Pour, par exemple, écrire une valeur dans le registre 0, le registre de données du port A, les connexions A11, A9 et A8 doivent être low. En écriture binaire, nous obtenons, pour l'octet fort du bus d'adresse, la valeur suivante:

A15	A14	A13	A12	A11	A10	A09	A08
1	1	1	1	0	1	0	0

Ce qui correspond à la valeur hexadécimale &F4.

Les 8 bits d'adresse inférieurs n'interviennent pas dans la sélection du 8255, une valeur entre &00 et &FF est ici possible.

Les bits mis dans l'octet fort ne sont pas non plus nécessaires en réalité à un adressage correct et on pourrait donc avoir l'idée d'utiliser comme octet fort la valeur 00H. Cela marcherait d'ailleurs. Mais comme le décodage des différents circuits intégrés périphériques se produit d'une semblable façon incomplète, les bits doivent être mis, sinon d'autres circuits intégrés tels que le CRTC ou le gate array pourraient se croire également appelés.

Mais revenons à notre exemple. Donc, pour charger une valeur dans le registre A, la valeur &F400 doit être placée sur le bus d'adresse. Ceci peut être obtenu avec les instructions:

```
LD    A,valeur
LD    BC,&F400
OUT   (C),A
```

Le registre de port C peut de même être lu avec les instructions:

```
LD    BC,&F600
IN    A,(C)
```

Les trois ports sont utilisés essentiellement en mode 0. Les 24 canaux d'entrée/sortie sont ainsi disponibles.

Le port A (&F400) est relié aux 8 canaux de données du générateur de son AY-3-8912. Suivant l'action demandée, le port A est programmé comme entrée ou sortie.

S'il est programmé en sortie, les instructions de commande sont envoyées au chip sonore à travers les 8 canaux du port. Vous trouverez le détail de ces instructions de commande dans le chapitre sur la programmation du AY-3-8912. Indiquons simplement pour le moment que le chip sonore dispose également d'un port 8 bits bidirectionnel. Une page de la matrice du clavier est connectée sur ce port. A travers le port A du 8255, il est possible par un détour du port du AY-3-8912 de savoir si une touche est enfoncée. A cet effet, le port A doit bien sûr être programmé en entrée.

Le port B (&F500) est programmé comme port d'entrée. Toutes les interrogations évoquées, hormis celle du clavier, se produisent à travers ce port. Les différents bits de ce port reçoivent l'affectation suivante:

Bit 0 : Ce bit interroge l'état du V Sync du CRTC. Comme cette interrogation doit aller très vite, le bit 0 peut être décalé dans le flag carry par simple rotation de la valeur lue avec INP. Il est ainsi possible de connaître rapidement l'état de V Sync.

Bits 1-3 : Ce bit est relié au pont LK4. Si ce pont est ouvert, le contrôleur vidéo est programmé pour le travail en PAL en 50 Hertz. Un pont fermé entraîne une programmation du CRTC pour la norme SECAM de 60 Hertz pour la fréquence de renouvellement de l'image. Cette possibilité de programmation différente est importante lorsque le CPC doit être utilisé à travers le module MP1 sur un téléviseur.

Bit 5 : Ce bit interroge l'état du signal EXP du connecteur d'extension.

Bit 6 : Ce bit restitue l'état d'une imprimante connectée. Comme l'imprimante ne peut pas recevoir de caractères en permanence, il est possible d'interdire un transfert de caractère en fixant cette connexion sur high.

Bit 7 : Les données fournies par le lecteur de cassette avec un niveau TTL sont lues à travers ce bit. Ici aussi vaut ce que nous disions pour le bit 0. Comme ce canal doit être examiné très rapidement, l'état de ce canal peut être déterminé très vite par une rotation unique du bit 7 vers le flag carry.

Le port C (&F600) est sur le CPC programmé comme port de sortie.

Quatre de ses huit canaux lui permettent de commander une partie de l'interrogation du clavier et deux autres bits sont utilisés pour le lecteur de cassette. Les deux bits restants sont employés pour la commande du chip sonore. Comme les canaux du port C peuvent être mis et annulés directement, celui-ci convient particulièrement à ce type de tâches.
Les différents bits sont ainsi utilisés:

Bits 0-3 : Ces bits commandent la matrice du clavier. Les quatre canaux programmés en sortie sont reliés à un décodeur BCD-décimal.
Ce décodeur met sur la masse une de ses 10 entrées, en fonction de l'information binaire en entrée. Les combinaisons en entrée autorisées sont les valeurs de 0 à 9.

Bit 4 : Ce bit commande le moteur du lecteur de cassette. Le moteur n'est cependant pas commandé directement, mais à travers un transisteur (et un relais commuté à la suite). Si ce bit est sur la masse, le moteur s'arrête.

Bit 5 : Les fréquences, qui doivent être reçues par le lecteur de cassette et qui produisent cette si douce mélodie, sont fournies par l'ordinateur à travers ce pin du 8255.

Bits 6-7 : Ces bits de port sont reliés aux connexions BC1 et BDIR du chip sonore et travaillent comme signaux de chip select et de strobe pour l'AY-3-8912. Vous trouverez une description plus détaillée de ces connexions dans le prochain chapitre sur le générateur de son.

1.8 Le générateur de son programmable AY-3-8912

L'AY-3-8912 de General Instruments est un générateur de son programmable (PSG) de grande classe. Il a été développé pour les jeux électroniques, afin de doter ceux-ci d'un son particulièrement réaliste alors que les premiers jeux électroniques ne pouvaient produire que des bruits vraiment monotones. Pour pouvoir être employé le plus universellement possible, le PSG a été doté d'un grand nombre de possibilités d'influencer le son. On pensa en outre lors du développement de ce circuit intégré que, dans pratiquement tous les domaines d'application, il faudrait pouvoir interroger des touches, joysticks ou commutateurs quelconques. C'est pourquoi on a donc également doté ce PSG d'un port parallèle 8 bits.

Les caractéristiques de ce circuit intégré sont les suivantes:

Trois oscillateurs de son programmables indépendamment
Un générateur de bruit programmable
Des sorties analogues entièrement commandées par logiciel
15 niveaux de volume étagés par logarithme
Courbes d'enveloppe programmables
Compatible TTL
Alimentation en courant continu de 5 Volts

L'AY-3-8912 dispose en tout de 16 registres, dont 15 registres peuvent être utilisés. A travers ces registres peuvent être programmées toutes les possibilités sonores du chip.

Le branchement du PSG peut être divisé en différents blocs de fonction.

Il y a d'abord le bloc des générateurs de son. Les générateurs de son reçoivent un signal d'horloge qui est produit à partir de la division par 16 du signal de l'horloge. Les générateurs de son sont responsables de la production fondamentale des trois fréquences de son carrées.

Le générateur de bruit produit un signal carré en modulation de fréquence dont l'écart de pulsation est influencé par un pseudo générateur de bruit.

Les mixeurs couplent les signaux de sortie des trois générateurs avec le signal de bruit. Le couplage peut être programmé séparément pour chaque canal.

Le bloc de fonction du contrôle d'amplitude offre deux possibilités à l'utilisateur. D'une part l'amplitude de sortie (le volume) des trois canaux peut être influencée à travers la programmation du registre de volume correspondant.

D'autre part il est possible de les faire influencer de façon variable par le PSG. La sortie du registre de courbe d'enveloppe est alors utilisée pour influencer le volume. Comme la courbe d'enveloppe peut être programmée avec quatre paramètres distincts, les possibilités d'influencer le son sont variées.

Le bloc de fonction du convertisseur D/A est responsable de la production du volume des signaux de sortie. Comme les informations de volume et d'enveloppe sont sous forme de valeurs digitales, elles sont converties dans le convertisseur D/A.

Le dernier bloc de fonction n'a rien à voir avec la production du son. Dans ce bloc sont placés deux ports I/O. Si vous êtes maintenant un peu surpris, c'est que vous nous avez lu attentivement. En effet le chip AY-3-8912 contient deux ports I/O complets dont un seul cependant est branché sur les pins de connexion. Le même chip est utilisé dans l'AY-3-8910, sur lequel les deux ports peuvent être utilisés.

1.8.1 Les connexions du chip sonore

Comme les noms des connexions du PSG ne sont pas suffisamment explicatifs, voici une description détaillée de la fonction des pins:

DA0 - 7 : Ces connexions du chip sonore sont reliées au bus de données du processeur. Le nom DA indique que aussi bien des Données que des Adresses (de registre) passent à travers ces connexions.

A8 : Cette connexion peut être comprise comme un signal CHIP-SELECT. Pour appeler des registres du PSG, ce signal doit être high.

BDIR &
BC1,2 : La connexion signal BDIR (Bus DIRection) et les connexions BC1 et BC2 (Bus Control) commandent l'accès aux registres sur le PSG. Au premier abord, l'affectation indiquée par le tableau peut paraître curieuse. Mais comme ce circuit intégré fut à l'origine développé comme composant du processeur 1610, un processeur 16 bits spécial de General Instruments, on a pris en compte lors de la conception les propriétés spéciales et des connexions de commande de ce processeur.

BDIR	BC2	BC1	Fonction du PSG
0	0	0	INACTIVE
0	0	1	LATCH ADRESS
0	1	0	INACTIVE
0	1	1	READ FROM PSG
1	0	0	LATCH ADRESS
1	0	1	INACTIVE
1	1	0	WRITE TO PSG
1	1	1	LATCH ADRESS

Dans ce tableau, seules quatre des huit combinaisons ont vraiment un sens. C'est pourquoi la connexion BC2 est souvent mise sur +5 Volts. Le tableau restant n'est donc plus influencé que par les signaux BDIR et BC1 et il se présente ainsi:

BDIR	BC1	Fonction du PSG
0	0	INACTIF, le bus de données du PSG a une valeur en ohm haute
0	1	READ, des données peuvent être lues dans les registres du PSG
1	0	WRITE, des données peuvent être écrites dans le registre du PSG sélectionné
1	1	LATCH, le numéro ou l'adresse du registre du PSG souhaité est écrit dans le PSG

ANALOG A C'est la sortie du canal A. Ici peuvent être retirés les sons produits par le canal A. La tension maximale en sortie est d' 1 Vss.

ANALOG B Fonction identique au pin 1, pour le canal B

ANALOG C Fonction identique au pin 1, pour le canal C

IOA7 - 0 : Les connexions IOA représentent le port 8 bits du PSG. Suivant la façon dont elles sont programmées, les connexions travaillent comme sortie ou entrée. Mais on ne peut fixer qu'un même mode de travail pour tout le port. On ne peut avoir simultanément des bits travaillant en entrée et d'autres en sortie.

CLOCK : De la fréquence de ce signal sont dérivées par division toutes les fréquences de son. La fréquence de ce signal devrait être entre 1 et 2 MHz.

RESET : Un niveau low sur cette connexion annule les valeurs de tous les registres. Sans reset, les registres contiennent aprés la mise sous tension des valeurs aléatoires dont la conséquence serait un bruit probablement très peu musical.

TEST1 : Test1 n'est utilisé que par le constructeur et ne doit pas être connecté en travail normal.

Vcc : Une tension de +5 Volts est placée sur cette connexion.

Vss : Ceci est la connexion de masse du PSG.

CHANNEL C (1)	DA 0
TEST 1	DA 1
Vcc	DA 2
CHANNEL B	DA 3
CHANNEL A	DA 4
Vss	DA 5
IOA 7	DA 6
IOA 6	DA 7
IOA 5	BC 1
IOA 4	BC 2
IOA 3	BDIR
IOA 2	A 8
IOA 1	RESET*
IOA 0	CLOCK

1.8.1.1 CHIP SONORE AY-3-8912

1.8.2 La fonction des différents registres du 8912

Comme nous avons maintenant vu comment les registres peuvent être appelés fondamentalement à travers les connexions BDIR et BC1, nous allons étudier quelles sont les fonctions remplies par ces registres. Le numéro de registre utilisé dans la liste suivante est identique au numéro qui doit être placé dans le registre d'adresse pour appeler le registre souhaité.

Il est un fait intèressant qui est que le registre d'adresse conserve son contenu jusqu'à ce qu'il soit à nouveau programmé. On peut donc accéder sans problème plusieurs fois successives à un registre de données, sans devoir chaque fois recharger le registre d'adresse.

Mais voici maintenant la description des registres:

Reg 0,1 : Ces registres déterminent la période et donc la fréquence du signal de son sur ANALOG A. Mais les 16 bits ne sont pas tous utilisés. Tous les 8 bits du registre 0 et les quatre bits inférieurs du registre 1 sont utilisés. La fréquence peut être influencée de façon fine avec le registre 0 ou grossièrement avec le registre 1. Plus la valeur 12 bits de ces registres est petite, plus le son est haut.

Reg 2,3 : Fonction comme Reg 0,1 mais canal B.

Reg 4,5 : Fonction comme Reg 0,1 mais canal C.

Reg 6 : Ce registre influence le générateur de bruit avec ces 5 bits inférieurs.

Reg 7 : Dans ce registre multi-fonctions, les différents bits contrôlent des tâches différentes, comme le montre le tableau suivant:

Bit 0 : mettre/couper le son du canal A 0=mis/1=non
Bit 1 : mettre/couper le son du canal B 0=mis/1=non
Bit 2 : mettre/couper le son du canal C 0=mis/1=non
Bit 3 : mettre/couper le bruit du canal A 0=mis/1=non
Bit 4 : mettre/couper le bruit du canal B 0=mis/1=non
Bit 5 : mettre/couper le bruit du canal C 0=mis/1=non
Bit 6 : Port A comme entrée/sortie 0=entrée/1=sortie
Bit 7 : Port A comme entrée/sortie 0=entrée/1=sortie

Reg 8 : Ce registre détermine le volume du signal sur le canal A. Les quatre bits inférieurs sont utilisés pour fixer le volume.
Le bit 4 a une signification particulière. S'il est mis, le volume est déterminé par le registre de courbe d'enveloppe et le contenu des bits 0 à 3 est alors ignoré.

Reg 9 : Comme Reg 8 pour le canal B

Reg 10 : Comme Reg 8 pour le canal C

Reg 11,12 : Les 16 bits de ces deux registres influencent la période de la courbe d'enveloppe. Le contenu du Reg 11 est considéré comme low byte, c'est-à-dire qu'il influence la période par étapes fines, alors que le Reg 12 est le high byte du générateur de courbe d'enveloppe.

Reg 13 : Les bits 0 à 3 de ce registre déterminent la forme de la courbe du générateur de courbe d'enveloppe. Il est presque impossible de rendre compréhensible par des mots l'affectation de ces bits. C'est pourquoi les courbes d'enveloppe sont montrées dans le graphique 1.8.2.1.

REG. 15			
B3	B2	B1	B0
CONTINUE	ATTACK	ALTERNATE	HOLD
0	0	–	–
0	1	–	–
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

1.8.2.1 Les courbes d'enveloppe du PSG

1.8.3 Le fonctionnement de l'AY-3-8912 sur le CPC

Nous allons nous intéresser dans cette section à la connexion concrète et certaines choses plus concrètes pour l'utilisation du chip sonore sur le CPC. Comme la description des registres qui précède était nécessairement abstraite et peut-être pas très aisément compréhensible, vous comprendrez mieux, après avoir lu ce chapitre, certaines particularités du PSG.

Jetons d'abord un coup d'oeil sur le schéma de fonction.
Le PSG y figure comme IC 102.
Les pins 3, 17 et 19 sont sur +5 Volts. L'AY-3-8912 reçoit son alimentation électrique à travers le pin 3. Comme BC2 (pin 19) et A8 (pin 17) sont sur +5 Volts, ils n'interviennent pas dans la sélection des registres.

Les connexions de commande des registres restantes BC1 (pin 20) et BDIR (pin 18) sont reliées aux bits de port PC6 et PC7 du 8255. Suivant l'état de ces connexions, des adresses de registre peuvent être communiquées au PSG ou des données peuvent être écrites ou lues dans le PSG.

Le transfert d'adresse et de données proprement dit se produit à travers les connexions D0 à D7 du PSG qui sont reliées au port A du 8255. Suivant l'action demandée, le port A doit être programmé comme entrée ou sortie.

Le signal de l'horloge sur le pin 15 est un signal carré d'une fréquence de 1 MHz. Ce signal est fourni par le gate array par division de la fréquence quartz. De ce signal sont dérivées par division de fréquence toutes les fréquences de son et de courbe d'enveloppe.

Le port I/O du PSG est relié au clavier et à la connexion pour le joystick. Vous trouverez dans un prochain chapitre une description détaillée du clavier et du joystick, nous ne nous intéressons ici qu'aux possibilités sonores du chip sonore.

Les connexions les plus importantes de ce circuit intégré sont certainement les trois sorties analogues A, B et sur les pins 1, 4 et 5. Ces sorties fonctionnent comme ce qu'on appelle des sorties Open-Emitter. Pour pouvoir sortir une tension alternative du son, des résistances sont nécessaires qui commutent entre sortie et masse. C'est la fonction des résistances R121, R122 et R123.

Le signal sonore est mixé par ces trois résistances à travers trois résistances et il se présente alors sous forme d'un signal mono sur la connexion 1 du port d'extension. Ce signal mono est cependant également conduit sur la connexion 1 du port d'extension. De là, ce signal arrive à l'amplificateur et au haut-parleur internes.

Les trois sorties sont cependant en outre conduites également vers la prise stéréo à l'arrière de l'ordinateur. A cet effet, le signal du canal B est envoyé de façon identique sur les deux canaux stéréo, à travers deux résistances. Les sorties A et C sont chacune envoyées directement sur un des canaux stéréo, à travers un condensateur de découplement.

Ce type de branchement rend même possible, avec une habile programmation, d'obtenir de véritables effets stéréo. Il serait par exemple imaginable de ne sortir d'abord un son que sur le canal A. Au bout de quelque temps, le même son pourrait être sorti en plus sur le canal B. On pourrait, ce faisant, faire monter lentement le volume du signal sur le canal B, alors que le volume du signal serait par contre réduit de façon symétrique. Le résultat serait qu'il semblerait que le son se promène d'un coin de la pièce vers le milieu entre les deux baffles. De là, il peut si nécessaire continuer vers l'autre coin.

Ces effets sont mêmes possibles en Basic avec la puissante instruction sound. Le manuel d'utilisation comporte cependant des contradictions dans l'indication de la répartition des trois canaux de son sur les deux canaux stéréo. Observez-le après avoir relié votre CPC à une chaîne stéréo. Seuls les sons du canal B apparaissent sur les deux canaux de la chaîne stéréo.

Mais comment le PSG produit-il au fond les sons? Examinons un peu comment les choses se produisent en détail sur un canal.
Comme nous l'avons déjà indiqué, tous les sons sont dérivés du signal de l'horloge sur le pin 15. Le signal d'horloge est d'abord divisé par 16. Il en résulte sur le CPC une fréquence de commande de 62,5 KHz. Cette fréquence est alors conduite vers un diviseur de fréquence programmable. Suivant le contenu des registres du générateur de son, la fréquence de commande est ou non à nouveau divisée, pour obtenir la fréquence voulue.

Les développeurs de ce circuit intégré ont à cet égard fait montre de beaucoup d'astuce. La chaîne de division n'est pas seulement constituée de flip-flops qui peuvent diviser la fréquence par deux. Par une technique de branchement spéciale, des facteurs impairs de division sont également possibles. La fréquence de commande peut tout-à-fait être divisée par 3 ou par 17. C'est uniquement ainsi que toutes les valeurs nécessaires peuvent être produites dans la zone de fréquences élevées.

Le contenu des registres du générateur de son détermine donc le facteur de division pour le signal sonore. Si le registre 0 du PSG reçoit la valeur 100, le registre 1 la valeur 0, la fréquence de commande sera divisée par 100. Sur la sortie de la chaîne de division du canal A se trouve un signal d'une fréquence de 625 Hertz.

Ce signal ne peut cependant pas encore être retiré sur la sortie A. Il faut d'abord que le canal correspondant soit activé. Ceci est obtenu en annulant le bit correspondant du registre 7. Comme nous avons choisi dans notre exemple le canal A, nous devons annuler le bit 0. Mais il faut, ce faisant, considérer l'état des autres bits. Sur le CPC, cela signifie concrètement qu'il ne faut pas modifier le bit 6 involontairement car sinon le clavier est bloqué.

Mais pour le moment on ne peut entendre encore aucun son, parce que le volume de chaque canal doit être fixé. Pour le canal A, c'est le registre 8 qui est responsable. Une valeur de 1 ne produit qu'un son très doux, alors qu'une valeur de 15 donne le volume maximal.

Si nous mettons (sur 1) le bit 4 du registre de volume, les informations contenues dans les bits 0 à 3 seront ignorées. Ce sont maintenant les registres 11, 12 et 13 qui déterminent le volume. Le volume n'est plus alors fixé sur une valeur mais il devient variable.

Considérons d'abord le registre 13. Ce registre porte le nom officiel de 'ENVELOPE SHAPE/CYCLE CONTROL REGISTER'. Sa fonction sera illustrée plus aisément grâce à un exemple.
Après que nous ayons fourni les valeurs adéquates aux registres 0, 1, 7 et 8, écrivons donc dans le registre 13 la valeur 12. Les bits 2 et 3 sont maintenant mis, alors que les 2 bits inférieurs sont annulés.
Le tableau fourni dans la description des registres montre pour cette combinaison une suite de dents montant lentement et retombant rapidement. En pratique, cela signifie que le volume du son monte tout d'abord lentement jusqu'au maximum. Puis le son est coupé et le volume recommence à monter. Cet état demeure jusqu'à ce qu'une nouvelle instruction soit envoyée au registre 13.

La durée de la montée du volume peut être fixée à travers les registres 11 et 12. Ces registres influencent de façon analogue aux registres des générateurs de son une autre chaîne de division programmable sur le PSG. La chaîne de division reçoit un signal qui correspond au signal de l'horloge divisé par 256. Cela donne une fréquence de 3906,25 Hertz correspondant à une période d'environ 250 microsecondes.

Si une valeur 1 est écrite dans le registre 11 et une valeur 0 dans le registre 12 qui travaille comme high-byte, le volume du son est réellement conduit en 250 microsecondes de 0 jusqu'au volume maximum. Cela figure cependant déjà dans la zone des sons audibles et produit un sifflement net qui est superposé au son véritablement souhaité.

C'est pour cette raison que les valeurs de registre choisies sont toujours nettement plus élevées. Avec la valeur maximale (255 dans Reg 11 et Reg 12), la montée jusqu'au volume maximum dure 16,8 secondes.

L'altération du volume à travers le registre d'enveloppe n'est pas utilisée par le logiciel du CPC. L'instruction ENV influence le volume du son uniquement à travers des manipulations des autre bits inférieurs du registre de volume. L'instruction ENT du CPC n'a pas d'équivalent sur le PSG. Cette fonction est produite par une modification habile des registres du générateur de son.

1.9 Le lecteur de disquette sur le CPC 664 et le CPC 6128

Contrairement à leur prédécesseur, le 464, le 664 et le 6128 disposent d'une interface lecteur de disquette et d'un lecteur de disquette intégrés dans leur boîtier. Cela ne rend pas seulement l'expansion slot à l'arrière mieux accessibles pour d'autres périphériques. Cela réduit également la place nécessaire pour travailler ainsi que le nombre de 'fils'.

Les branchements sont fondamentalement compatibles sur les trois ordinateurs, aussi bien pour ce qui concerne leur fonctionnement que pour ce qui concerne le logiciel. Toute la littérature disponible sur les lecteurs de disquette AMSTRAD (par exemple, le livre du lecteur de disquette AMSTRAD, Data Becker - Micro Application) s'applique donc aux trois modèles d'ordinateur, de sorte que les possesseurs d'un CPC 664 ou d'un CPC 6128 peuvent également profiter des livres déjà parus.

Le point central du controller board est constitué par le floppy disk controller (FDC) uPD 765. Ce circuit intégré constitue l'interface entre les lecteurs et le processeur du CPC. On peut certes tout à fait construire des lecteurs de disquette sans FDC mais la grande 'intelligence propre' du FDC simplifie grandement la construction. L'électronique nécessaire ainsi que l'importance du logiciel d'exploitation sont considérablement réduites par l'emploi d'un FDC. Un exemple éclairera ce point.

Le lecteur de disquette 1541 de la firme Commodore que beaucoup d'entre vous connaissent certainement comme lecteur de disquette du Commodore 64 est un lecteur de disquette construit sans FDC. Sans compter la lenteur de la transmission des données dée à la construction même du lecteur (lenteur qui ne peut que faire sourir les possesseurs d'un CPC), l'investissement électronique pour ce lecteur est beaucoup plus important que sur le lecteur de disquette du CPC. L'électronique digitale du 1541 contient un processeur propre, des circuits intégrés périphériques de 40 pôles et une masse de circuits intégrés TTL de toute sorte. Cette masse de composants correspond à celle que requiert un CPC 664 complet!

Le logiciel d'exploitation du 1541 est, avec 16 K, deux fois plus grand que l'AMSDOS. Il est évident que les développeurs (pour des raisons de confort) et les commerçants (pour des raisons de coût) recourent volontiers aux FDCs dont l'utilisation est si pratique.

L'électronique compléte du controller se compose de 12 circuits intégrés, 2 transistors, 20 résistances, 15 condensateurs et une diode. Ce petit nombre de composants n'a pu être obtenu que par la haute intégration de trois circuits intégrés. Il s'agit du FDC, du séparateur de données et de la ROM avec l'AMSDOS.

La ROM AMSDOS, d'une taille de 16 K-octets contient, dans envirion 8 K octets, toutes les routines essentielles qui sont nécessaire pour gérer le lecteur de disquette. Cette ROM contient en outre dans son logement à 28 pôles, dans les 8 K restants, une partie de l'interpréteur LOGO fourni sur la disquette CP/M 2.2.

1.9.1 LE FDC 765

Le FDC exploité par les firmes NEC sous le nom de uPD765, ROCKWELL sous le nom de R 6765 et INTEL sous le nom de 8765, peut être considéré comme un microprocesseur très spécialisé. Les possibilités de ce circuit intégré sont si étendues et si complexes que ce qualificatif n'est certainement pas exagéré.

Le format de données utilisé par le FDC correspond au format IBM 3740 en densité simple et au format IBM System 84 en double densité. De ce fait, les disquettes Commodore ou Apple par exemple ne peuvent malheureusement pas être lues ni écrites.

Avec ces 40 pins, il fournit tous les signaux nécessaires pour exploiter les lecteurs du marché des tailles 8", 5 1/4" et 3". Les signaux de commande disponibles permettent au développeur de connecter ce FDC à presque n'importe quel processeur. Deux possibilités fondamentales de connexion et d'exploitation sont offertes. La première méthode est l'exploitation DMA. En liaison avec un DMA controller, le FDC peut prendre en charge le contrôle de la mémoire du système informatique pour le transfert de données en lecture et en écriture.

Il retire alors de la mémoire, à l'aide du DMA controller, les nouvelles données nécessitées ou écrit dans la mémoire, également en contournant le processeur, les données lues sur la disquette. Cette très rapide méthode de transfert de données n'est cependant pas utilisée sur le CPC et nous ne l'avons évoquée que par souci d'exhaustivité.

Avec la seconde méthode, celle utilisée sur le CPC, le transfert de données est pris en charge par le processeur. Pour cette seconde méthode, il faut cependant à nouveau distinguer entre deux possibilités d'exploitation du FDC.

Il y a d'abord la méthode des interruptions. Pour chaque transfert de données, une interruption est alors produite. Dans la routine d'interruption du processeur doit alors être fourni ou lu par le processeur le prochain octet de donnée ou d'instruction. Du fait de la structure électronique du CPC, il ne pouvait non plus être question de cette méthode, de sorte que les développeurs ont choisi la méthode polling. Le processeur doit alors examiner réguliérement dans les registres du FDC quelle est la prochaine action demandée par le FDC.

Mais considérons tout d'abord un aperçu des données techniques du 765. Gardez cependant à l'esprit que les développeurs du controller board n'ont pas utilisé toutes les possibilités du 765.

* longueur de secteur programmable
* toutes les données du lecteur programmables
* jusqu'à quatre lecteurs connectables
* transfert de données au choix, en mode DMA ou pas en mode DMA
* connectable à presque tous les types de processeur courants
* alimentation électrique simple 5 volts
* horloge monophase simple de 4 ou 8 MHz
* habitacle de 40 pôles du circuit intégré

Nous allons maintenant nous intéresser un peu plus en détail au dernier point de cette bréve présentation.

1.9.2 L'AFFECTATION DES CONNEXIONS DU FDC

Les connexions du FDC 765 peuvent être subdivisées en plusieurs groupes. Le premier groupe de connexions représente l'interface avec le processeur systéme. C'est donc à travers ces connexions que le FDC est commandé par le processeur.

Le deuxième groupe n'est nécessaire qu'en liaison avec l'exploitation DMA. C'est à travers ces signaux que communiquent le DMA controller et le FDC.

L'interface avec les lecteurs de disquette est constituée par le troisième groupe, qui est avec 19 connexions le groupe le plus important en nombre.

Dans le quatrième et dernier groupe peuvent être regroupées les connexions pour l'alimentation électrique et l'horloge.

Commençons l'examen des connexions par le premier groupe, l'interface avec le processeur.

L'interface avec le processeur

RESET : L'entrée RESET du FDC est active high. En exploitation normale, cette connexion est placée sur masse. Un high sur le pin RESET place le FDC dans un état déterminé.

CS* : CHIP SELECT. Un low sur ce pin sélectionne le FDC. Ce n'est qu'avec CS* = low que RD* et WR* deviennent valables pour le FDC. Comme la production du CS est à la libre disposition du développeur, le FDC peut être appelé au choix Memory-Mapped, donc comme élément de la zone de la mémoire, ou à travers des adresses de port.

RD* : READ*. Cette connexion doit être reliée au signal RD* du processeur. Dés que le processeur veut lire des données à partir du FDC, ce canal est mis sur low.

WR* : WRITE*. De même que le canal RD* signale des accés en lecture du processeur, un low sur WR* indique que le processeur écrit des données ou des instructions dans le FDC.

A0 : ADRESS LINE 0. Le FDC ne dispose que de deux adresses pouvant être appelées de l'extérieur. La distinction entre les deux adresses est effectuée avec le signal A0. Ce canal est normalement relié au bit d'adresse le plus bas du processeur.

Pin	Left	Right	Pin
1	RESET	VCC (+5V)	
	AD	RW/SEEK	
	WR	LCT/DIR	
	CS	FLTR/STEP	
	A 0	HOLD	
	DB 0	READY	
	DB 1	WPRT/2 SIDE	
	DB 2	FLT/TRKO	
	DB 3	PS 0	
	DB 4	PS 1	
	DB 5	WDATA	
	DB 6	US 0	
	DB 7	US 1	
	DRQ	SIDE	
	DACK	MFM	
	TC	WE	
	INDEH	SYNC	
	INT	RDATA	
	Ø	WINDOW	
	GND	WCLK	

1.9.1.1 PINOUT du FDC 765

DB0 - DB7 : DATABUS 0-7. Ces connexions du FDC sont reliées au bus de données du systéme. Toutes les instructions et données sont transportées à travers ces huit connexions bi-directionnelles. La direction des données est chaque fois déterminée soit par le processeur, soit par le DMA controller en mode DMA.

INT : INTERRUPT. A travers cette connexion, le FDC peut produire une interruption du processeur du systéme. Les interruptions sont produites à chaque transfert d'octet (non connecté sur le CPC).

Signaux pour le mode DMA (inutilisé sur le CPC)

DRQ : DMA REQUEST. A travers cette connexion, le FDC signale au DMA controller qu'un accés à la mémoire doit se produire. A la prochaine occasion possible, le DMA controller prendra en charge le bus systéme. Le processeur est alors déconnecté.

DACK* : DMA ACKNOWLEDGE. Ce signal indique au FDC que le DMA controller a pris en charge le bus et a maintenant commencé le transfert de données.

TC : TERMINAL COUNT. Un niveau high sur cette connexion interrompt le transfert de données vers et à partir du FDC. Bien que cette connexion soit essentiellement utilisée en mode DMA, le transfert de données peut également être interrompu à travers cette connexion dans les systémes commandés par interruption.

L'interface disquette

US0, US1 : UNIT SELECT 0/1. A travers ces deux connexions peuvent être connectés directement deux lecteurs de disquette, mais avec l'aide d'un décodeur deux-à-quatre, ce sont même quatre lecteurs qui peuvent être connectés.

C'est à travers ces connexions qu'est appelé chaque fois le lecteur voulu pour l'écriture ou la lecture de données.

HD : HEAD SELECT. Comme le FDC est conçu pour l'exploitation de lecteurs de disquette à double tête de lecture, la sélection de la tête peut s'effectuer à travers cette connexion lorsqu'on utilise de tels lecteurs.

HDL : HEAD LOAD. Ce signal est employé presque exclusivement sur les lecteurs 8". Les moteurs de ces lecteurs ne sont pas mis en marche quand c'est nécessaire, mais ils tournent normalement sans arrét. Mais pour ménager malgré tout la disquette et la tête d'écriture, la tête n'est normalement 'chargée', à travers un aimant qui l'améne prés de la surface de la disquette, que lorsque c'est nécessaire. La commande de l'aimant s'effectue alors au moyen de HDL.

IDX : INDEX. Le signal produit par le faisceau lumineux est placé sur cette connexion. Il signale au FDC le début physique d'une piste.

RDY : READY. Le signal READY fourni par le lecteur de disquette indique qu'une disquette se trouve dans le lecteur de disquette et que celle-ci tourne à une vitesse minimum déterminée. Ce n'est qu'aprés apparition du READY que le FDC accède au lecteur de disquette.

WE : WRITE ENABLE. Cette sortie du FDC doit être high pour que des données puissent être écrites sur la disquette.

RW/SEEK : READ WRITE/SEEK. Un lecteur de disquette produit au total plus de signaux qu'il n'y en a de disponibles pour l'interface disquette sur un socle de 40 pôles.

Toutefois, tous les signaux ne sont pas nécessaires en même temps à tout moment. Huit de ces signaux disquette ont été pour cette raison séparés en deux groupes qui peuvent être placés de façon sélective sur quatre connexions du FDC. Le FDC sélectionne de lui-même à travers la connexion RW/SEEK les signaux dont il a besoin à un moment donné.

FR/STP : FIT RESET/STEP. C'est le premier des quatre doubles signaux du FDC. Cette sortie a différentes significations suivant l'opération exécutée. D'une part cette connexion permet de restaurer le flip-flop d'erreur qui existe sur certains lecteurs. La seconde utilisation, beaucoup plus courante est la commande de l'entrée des pas du lecteur. A chaque déplacement de tête, les impulsions nécessaires sont fournies sur cette connexion.

FLT/TR0 : FAULT/TRACK0. Cette entrée peut elle aussi évaluer deux signaux différents. Si une opération SEEK (voir Programmation du FDC) est exécutée, le signal track0 du lecteur est attendu sur cette connexion. Ce signal est produit par un faisceau lumineux ou par un commutateur mécanique, lorsque la tête de lecture/écriture se trouve sur la piste physique 0. Le seconde fonction, le signal Fault est générée par certains lecteurs en cas d'erreur. Elle peut être à nouveau annulée par le FDC avec le signal FR/STP défini plus haut. Ce signal est contrôlé lors d'opérations Read/Write du FDC.

LCT/DIR : LOWCURRENT/DIRECTION. Les impulsions de pas de FR/STP indiquent bien sûr uniquement que la tête doit être déplacée. LCT/DIR détermine alors en mode Seek la direction du déplacement de la tête. La fonction LOW CURRENT est nécessaire lors de l'écriture des données. Ce signal permet de diminuer le flot d'écriture sur les pistes intérieures.

Vous trouverez des détails sur ce signal dans la description des bases théoriques de la sauvegarde sur disquette.

WP/TS : WRITE PROTECT/TWO SIDE. Indépendamment des diverses méthodes utilisées avec les différentes tailles de lecteurs de disquette, l'état de protection contre l'écriture est communiqué par le lecteur de disquette au controller, sous forme d'un signal. Ce signal est testé par l'entrée WP/TS lors des opérations de lecture/écriture. Le signal TS est testé lors des opérations Seek. Il n'est nécessaire qu'en liaison avec les lecteurs à double tête de lecture.

WDA : WRITE DATA. Les données sérielles à écrire sont transmises à travers cette connexion au lecteur de disquette. Ce peuvent être aussi bien les données pour l'écriture d'un secteur que toutes les informations nécessaires lors du formatage.

PS0, 1 : PRE SHIFT 0/1. A travers ces connexions, le FDC indique pour le format à double densité (MFM) à une électronique appropriée comment le flot sériel de données doit être écrit sur la disquette. Les trois états possibles sont EARLY, NORMAL et LATE pour la précompensation.

RD : READ DATA. Les informations lues sur la disquette sont entrées dans le FDC à travers cette entrée. C'est à partir de ce flot sériel de bits que les octets écrits à l'origine sont reconstitués.

RDW : READ DATA WINDOW. Ce signal est obtenu dans un séparateur de données à partir des données lues. Plus de détails dans le chapitre Bases de la sauvegarde sur disquette.

VCO : VCO SYNC. Ce signal est nécessaire pour la commande du VCO dans le séparateur de données PLL.

MFM : MFM MODE. Cette connexion signale si le controller travaille en format simple densité (MF) ou double densité (MFM).

Alimentation électrique et signaux d'horloge

Vcc : +5 Volts. C'est à travers cette connexion que le FDC reçoit son alimentation en courant électrique. La tension de 5 Volts doit être constamment dans la zone de +-5 %. L'intensité de courant nécessaire est au maximum de 150 mA.

GND : GROUND. Connexion à la masse du FDC.

CLK : CLOCK. Le FDC a besoin d'une fréquence d'horloge. Suivant les lecteurs, cette fréquence doit être de 4 MHz (pour les 5 1/4 et les formats plus petits) ou de 8 MHz (pour les 8").

WCK : WRITE CLOCK. La fréquence de ce signal doit être sélectionnée en fonction du format de données choisi. Pour MF, la fréquence doit être de 500 kHz, pour MFM, de 1 MHz. Cette fréquence détermine la vitesse de transmission des données vers et à partir du lecteur de disquette.

1.9.3 L'EMPLOI DU FDC 765 SUR LE CPC

Malheureusement les développeurs du CPC sont loin d'avoir utilisé toutes les possibilités du FDC. C'est ainsi que deux lecteurs seulement peuvent être connectés au lieu des quatre possibles. L'exploitation de lecteurs à double tête n'est pas non plus possible car le signal HEAD SELECT, s'il est bien connecté, n'est cependant pas utilisé. Le sort du signal HEAD LOAD est encore pire puisqu'il n'est connecté nulle part.

Ce défaut est cependant le plus facile à admettre puisqu'une exploitation de disquette 8" est d'une part sans intérét pour l'utilisateur 'moyen' du fait des énormes dimensions physiques de ces lecteurs et qu'elle est d'autre part rendue impossible par d'autre détails dans les connexions du controller.

Malgré ces réserves, le controller a été très intelligemment construit pour le but recherché, l'exploitation sans problème de deux lecteurs 3". Avec une économie maximum d'électronique, un controller a été réalisé qui présente d'excellentes caractéristiques techniques.

Malgré l'esprit d'économie des développeurs, on n'a heureusement pas limité la fiabilité de l'appareil. On a ainsi adapté comme 'auxiliaire' au FDC 765 un composant qui arrache aux experts en électronique, pour le moins, une moue d'approbation. Nous pensons au séparateur de données intégré, le SMC 9216 à 8 pôles qui est parfaitement approprié. Tous les signaux pour l'interface disquette du FDC, à l'exception du signal pour la mise en marche des moteurs du lecteur de disquette, sont produits par le FDC et le séparateur de données.

Bien que l'exploitation DMA représente la méthode la plus simple et la plus élégante pour connecter le disk controller, c'est une autre voie qui a été choisie, certainement pour des raisons de coût. Le processeur synchronise le transfert de données au vu du registre d'état principal. Les interruptions produites par le controller ne sont pas utilisées. Effectivement, la connexion d'interruption du FDC n'est pas branchée.

Le FDC est situé sur les adresses de port &FB7E et &FB7F. A la première adresse se trouve le registre d'état principal, la deuxième adresse appartient au registre de données.
Une troisième adresse est occupée par le Controller Board. Sur le port &FA7E se trouve un flip-flop à travers lequel les moteurs du lecteur de disquette sont commandés. Si on écrit un 1 sur ce port (OUT &FA7E,1 en Basic), les moteurs de tous les lecteurs connectés sont mis en marche, par contre si on écrit un 0, tous les moteurs sont à nouveau arrétés.

1.10 Les interfaces du CPC

Le concept d'interface peut être défini comme un point de liaison entre l'ordinateur et le monde extérieur. Le monde extérieur peut être aussi bien un autre ordinateur qu'une imprimante ou un autre périphérique, qu'un appareil de mesure ou un homme. D'après cette définition du monde extérieur, nous ne décrirons pas seulement dans ce chapitre les connexions figurant à l'arrière de l'ordinateur mais également le clavier, la connexion du moniteur et le lecteur de cassette.
Les interfaces les plus importantes pour l'utilisateur sont le clavier et le moniteur car celles-ci représentent le contact immédiat avec l'ordinateur. Commençons donc par ces deux interfaces.

1.10.1 Le clavier

Le clavier du CPC comprend en tout 74 touches. Comme les deux touches SHIFT sont branchées parallèlement, il y donc 73 touches différentes qui peuvent être interrogées.
La matrice dans laquelle les touches sont rangées comprend 8 fois 10 canaux. Comme les joysticks peuvent également être interrogés à travers cette matrice, 79 positions de touche sont donc occupées en tout. Le second joystick connecté directement sur le premier n'est pas connecté à des positions autonomes de la matrice, les branchements correspondants sont parallèles à des touches du clavier.

Du point de vue électronique, le clavier est interrogé à travers le 8255 et le chip sonore. Cela fonctionne à peu près de la façon suivante.

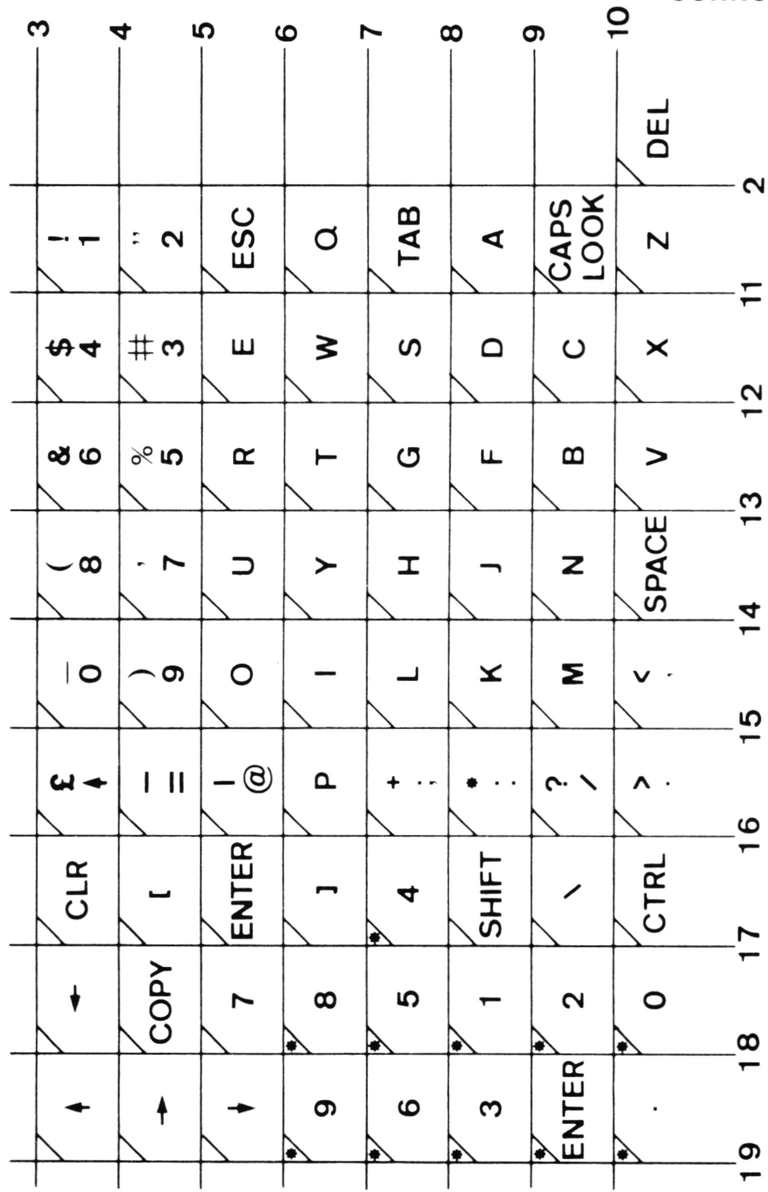
Le 8255 fournit aux sorties de port PC0 à PC3 une moitié d'octet qui est transformée par un décodeur 74LS145 en une information décimale. Suivant l'information figurant en entrée, une des dix sorties devient low. Ce décodeur est pour cette raison également appelé décodeur BCD-décimal. Si l'information en entrée n'est pas comprise entre 0 et 9, toutes les sorties du décodeur sont sur high.

Le port parallèle du chip sonore est programmé pour l'interrogation du clavier comme port d'entrée. Si aucun signal ne se trouve sur ces entrées, on obtient lors de la lecture du port un 1 sur toutes les entrées, en tout donc &FF.

Soit maintenant une information en entrée sur le décodeur de &04. La sortie pin 5 deviendra donc low. Mais le chip sonore ne le prendra pas en compte tant qu'aucune touche correspondante ne sera enfoncée. Le fait d'appuyer sur la touche ESC n'aura par exemple aucun effet à ce moment puisque la sortie pin 8 du décodeur est high. Mais si par contre la touche ESPACE est enfoncée, la valeur fournie par le chip sonore se modifiera. A cause de la touche enfoncée, le bit 7 du port est maintenant sur la masse et nous obtenons du chip sonore la valeur &7F.

Toutes les touches sont examinées 50 fois par seconde. A cet effet, les valeurs 0 à 9 sont sorties l'une après l'autre sur les quatre sorties utilisées du port C et la valeur du chip sonore est examinée après chaque sortie. Si des touches enfoncées sont alors enregistrées, les touches enfoncées sont placées dans un tableau et sont si nécessaire converties en numéros de touche et en caractères correspondants.

Un fait très pratique sur le clavier est que jusqu'à 20 caractères sont stockés provisoirement. Dans des programmes Basic, on peut déjà commencer à faire des entrées alors que l'ordinateur n'a pas terminé certains calculs ou qu'il est occupé à la sortie sur écran. L'interrogation du clavier n'est bloquée que lors de l'utilisation du lecteur de cassette, ainsi que lors du listage de programmes BASIC ou encore lors de certaines opérations avec la disquette, car il ne reste pas assez de temps pour cela, étant donné le timing très précis de ces opérations. La seule exception est la touche ESC qui est en effet nécessaire pour permettre une éventuelle interruption de l'opération en cours.



1.10.1.1 La matrice du clavier

Le clavier a par ailleurs une petite particularité. Essayez par exemple d'appuyer simultanément sur les touches J, K et L. De façon très surprenante, vous voyez apparaître en outre un H sur l'écran. Cela se produit toujours lorsque vous appuyez sur trois touches qui constituent les angles d'un carré dans la matrice du clavier, de même par exemple que 123 ou DFG. Dans ce cas apparaît simultanément le quatrième caractère de la matrice.
Ce 'défaut' est sans grande conséquence et vous pouvez par ailleurs également interrompre des programmes en appuyant simultanément sur les touches 2, 3 et E.

1.10.2 La connexion vidéo

La connexion vidéo du CPC fournit tous les signaux nécessaires au fonctionnement d'un moniteur. Il est à cet égard indifférent qu'il s'agisse du moniteur fourni avec ou de (presque) n'importe quel autre.

Le gate array fournit quatre signaux pour le moniteur. Trois signaux contiennent les informations sur la couleur, le quatrième signal est un mélange des signaux du CRTC V Sync et H Sync.
Ces signaux sont mixés avec des résistances et ils sont amplifiés par un transistor. Le signal de sortie ainsi produit est appelé LUM et sert aux moniteurs verts de signal vidéo. Mais également des moniteurs couleur courants peuvent être utilisés à travers ce signal pour représenter des couleurs.

1.10.3 La connexion du lecteur de disquette

Ce ne sont pas seulement les ordinateurs CPC, mais également les manuels livrés avec, qui sont plus complets et mieux faits que pour beaucoup de produits concurrents. Toutefois, quelques erreurs se sont glissées ici ou là dans ces manuels. Par exemple, les connexions situées à l'arrière du CPC 6128, sont représentées, dans le manuel d'utilisation, comme sur le 664. Cela n'est d'ailleurs gênant que pour la connexion du lecteur de disquette qui présente quelques différences, alors que les autres connexions sont identiques, de par leur nombre et leur affectation, sur les deux modèles.

La connexion du 664 est une plaque de conducteurs de 34 pôles alors que, sur le 6128, c'est une bague Centronics de 36 pôles. Sur cette douille, les connexions 1 et 19 ne sont pas affectées. Comme, dans les prises Centronics, les connexions sont comptées autrement que sur les plaques de conducteurs, la désignation des connexions ne correspond malheureusement pas aux indications fournies dans le manuel. La disposition physique des connexions est cependant identique à celle montrée dans le manuel. On peut aisément réaliser soi-même un adapteur pour un quelconque second lecteur de disquette, avec un morceau de câble plat à 34 pôles, une fiche Centronics et la fiche correspondant au lecteur voulu. Il est ainsi possible de connecter également des lecteurs de disquettes 5 pouces 1/4.
Nous ne décrirons pas ici à nouveau les connexions de l'interface disquette puisque cette description a déjà été faite dans le chapitre sur le contrôleur disquette.

1.10.4 Le lecteur de cassette

Bien que votre CPC possède déjà un lecteur de disquette intégré, l'ordinateur dispose d'une connexion pour le travail avec un lecteur de cassette. Cela vous permet d'une part d'utiliser les logiciels disponibles pour le CPC 464, et, d'autre part, la cassette est un moyen de stockage de données remarquable pour un prix très intèressant. Cette connexion permet également de préserver la compatibilité entre les différentes machines CPC.

Vous pouvez connecter n'importe quel modèle courant de lecteur de cassette. L'important est simplement qu'il y ait un niveau de signal suffisant sur la bague de l'écouteur et que le son du magnétophone ne soit pas trop "monotone". D'autre part, de trop grandes variations dans la vitesse d'écriture peuvent perturber le timing qui doit être respecté, surtout avec une vitesse de transmission élevée.

Mais venons-en au format d'écriture.
Le lecteur de cassette ne peut fondamentalement stocker les données, de même que le lecteur de disquette, que bit par bit.

Chaque octet à stocker doit donc être décomposé en ses différents bits et être transmis sous cette forme. Cette décomposition est réalisée par le processeur par logiciel, le bit supérieur étant à cet effet envoyé en premier au lecteur de cassette.

Le signal fourni par le 8255 pour le lecteur de cassette est un signal carré. Chaque bit est marqué par une vibration carrée, dans laquelle la phase low est exactement aussi longue que la phase high. On dit également que le signal carré a un rapport de 1:1. Un bit 0 nécessite moitié moins de temps qu'un bit 1.

C'est pourquoi les indications sur la vitesse d'écriture ne peuvent être que des indications imprécises. Il est évident qu'un bloc composé uniquement d'octets 0 sera sauvegardé en deux fois plus de temps qu'un bloc d'à peu près la même taille ne comportant que des &FF. Mais comme la répartition des bits 0 et 1 dans un bloc de données est à peu près égale, on peut s'en tenir aux indications de 1000baud(1baud=1bitparseconde)pourSUPER-SAVE(SPEEDWRITE 0) et de 2000 baud pour SPEED-LOAD (SPEED WRITE).

Chaque fichier cassette, qu'il s'agisse d'un fichier programme ou d'un fichier de données, peut comporter au maximum 65536 octets. Les fichiers sont écrits par blocs comportant chacun au maximum 2048 octets. Chaque bloc comprend au maximum huit segments de données de 256 octets. Devant chaque bloc est écrit un header, c'est-à-dire une tête de bloc.

Bien qu'il n'y ait pas de liaison électrique avec l'amplificateur et le haut-parleur, il est possible, lorsque le volume est suffisant, de suivre à l'oreille le chargement et la sauvegarde de données et de programmes.

Le header de bloc est facile à identifier à l'oreille. On entend en effet un long ton égal suivi de quelques octets qu'il n'est toutefois pas possible de distinguer à l'oreille.

Le ton long et égal est une série de 2048 bits 1. Après ces bits vient un seul bit 0 puis un octet de synchronisation.

La longue suite de bits 1 est nécessaire à l'ordinateur pour déterminer la vitesse (baud-rate). Le bit 0 indique à l'ordinateur que cette tête est terminée et l'octet sync est nécessaire pour distinguer entre l'information du header et les données.

L'information du header figure dans une zone de données longue de 64 octets qui est transmise devant chaque bloc de 2K de données. Dans ce header de fichier figurent les informations sur le fichier lui-même, par exemple le nom, si le fichier est ou non protégé, s'il s'agit d'un programme Basic ou d'un fichier Ascii et quelle est la longueur du programme.

Octets 0-15 : Nom du fichier, si moins de 16 octets, rempli avec 00

Octet 16 : Numéro de bloc, dans cet octet figure le numéro qui sera affiché lors du chargement ou également avec Catalog.

Octet 17 : Si dans cet octet figure une autre valeur que 00, il s'agit du dernier bloc du fichier.

Octet 18 : Cet octet contient le type de fichier. L'information est codée dans les différents bits. La signification des bits vient à la suite de ce tableau.

Octets 19,20 : Ces octets contiennent la longueur des informations du fichier. Si le bloc, donc les 2 K, est entièrement écrit, ces octets contiennent la valeur &0800. Dans le dernier ou unique bloc, figure ici le nombre d'octets du bloc.

Octets 21,22 : Ces octets indiquent l'adresse de chargement, à partir de laquelle les données ont été écrites à l'origine.

Pour les programmes Basic, c'est l'adresse décimale 368, pour les fichiers binaires, donc pour le langage-machine, c'est normalement l'adresse où tourne le programme en mémoire.

Octet 23 : Si le contenu de cet octet est différent de 0, il s'agit du premier bloc du fichier.

Octets 24,25 : Ces octets contiennent la longueur du fichier.

Octets 26,27 : Si un programme machine est lancé avec 'RUN "nom du fichier"', le contenu de ces octets du header sera interprété comme l'adresse de début d'un fichier en langage-machine. Le programme sera donc automatiquement lancé à l'adresse indiquée.

Les octets restants 28 à 63 du header ne sont pas utilisés par le système d'exploitation et sont à la disposition des programmeurs chevronnés.

Mais voici maintenant le décodage des bits de l'octet 18 du header:

Bit 0 : Si ce bit est mis, le fichier correspondant est déclaré protégé. Les programmes protégés peuvent être produits en Basic avec 'SAVE "NOM",p'.

Bits 1-3 : Ces bits déterminent le type de fichier. Bien que trois bits permettent 8 différents types de fichier, seuls les types de fichier programme Basic (0), fichier binaire (1) et fichier de données ascii (3) sont utilisés.

Bits 4-7 : Ces bits comportent normalement un 0, seuls les fichiers Ascii ont un 1 dans le bit 4.

Comme nous l'avons déjà indiqué, les informations stockées dans les différents blocs sont encore subdivisées en différents segments.

Chaque segment se compose de 256 octets de données et d'octets de check sum (contrôle du total). La check sum de chaque segment est calculée d'après une formule spéciale et permet de vérifier lors de la lecture du fichier si les bits ont été correctement transmis. Dès lors que la checksum calculée ne correspond pas aux valeurs lues, le READ ERROR B est affiché.

Le READ ERROR A indique qu'un bit a été lu dont la durée était trop longue par rapport aux valeurs calculées pour les bits nuls ou 1. Cette erreur se produit souvent, lors de la lecture de programmes, lorsque la cassette, qui coinçait lors de la sauvegarde, est maintenant fluide.

La troisième erreur possible est le READ ERROR D. Cette erreur ne devrait se produire que rarement car elle signale que le bloc lu est plus long que les 2048 octets autorisés. Cela ne peut toutefois se produire que si l'utilisateur écrit dans les informations du header, lors de la sauvegarde, des valeurs plus grandes que celles autorisées.

Vous connaissez certainement l'instruction Basic 'SPEED WRITE par'. Suivant les paramètres utilisés, les données sont stockées sur la cassette à une vitesse moyenne de 1000 ou 2000 baud. Ceci n'atteint cependant pas encore la vitesse la plus grande possible. Par l'utilisation d'une routine du système d'exploitation, la vitesse (baud rate) peut être fixée à toute valeur comprise entre 700 et environ 3600 baud. La routine nécessaire est à l'adresse &BC68. Elle attend des paramètres dans deux registres et fixe la vitesse d'écriture en fonction de ces paramètres. Une valeur est transmise à la paire de registres HL qui détermine la vitesse (baud rate). La formule pour déterminer cette valeur est:

Baud rate=333333/moitié de la longueur d'un bit nul

Cela donne pour 1000 baud une vitesse de 666 microsecondes pour un bit nul; un bit 1 dure exactement le double.

L'électronique utilisée dans le lecteur de cassette a cependant une particularité.

Si des bits nuls et des bits 1 sont lus tour à tour, l'électronique essaye de combler les différences de durée. Les bits 1 deviennent de ce fait plus courts, alors que les bits nuls apparaissent comme des impulsions plus longues qu'on ne l'aurait attendu après l'écriture. Pour cette raison, une compensation anticipée doit être exécutée et les bits nuls sont écrits plus brièvement, alors que les bits 1 sont écrits avec des durées légèrement plus longues. Ces durées nécessaires pour la compensation anticipée sont transmises à la routine dans l'accumulateur.

Pour des tentatives de fixer la vitesse d'écriture la plus rapide, qui est à moitié fiable, il suffit de transmettre dans l'accumulateur une valeur de 10. Pour écrire avec une vitesse de 3600 baud, il faut activer la routine suivante:

```
LD HL,93
LD A,10
CALL &BC68
RET
```

Ces quelques octets peuvent facilement être placés dans la mémoire avec les lignes suivantes:

```
10 MEMORY HIMEM - 10
20 FOR I = 1 TO 9
30 READ X : POKE HIMEM + I,X
40 NEXT I
50 CALL HIMEM + 1
60DATA&21,&5D,&00,&3E,&0A,&CD,&68,&BC,&C9
```

Ne craignez pas de faire varier quelque peu les valeurs dans HL et dans l'accumulateur (les deuxième et cinquième valeurs de la ligne de Data), pour déterminer la plus haute fréquence d'écriture possible. Elle dépend des cassettes utilisées. Mais les propriétés de rotation régulière de votre lecteur de cassette jouent également un rôle non négligeable.

Si les valeurs sélectionnées sont trop petites, le CPC ne peut plus alors tenir les durées réclamées et vous obtenez comme résultat le message d'erreur WRITE ERROR A.

Encore un conseil pour finir:
Vous avez certainement remarqué que lorsque vous sauvegardez de très longs programmes avec de nombreuses variables, cela peut durer jusqu'à 15 minutes jusqu'à ce que les données ou le programme soient sauvegardés. Cela vient du fait que le CPC nécessite pour la sauvegarde une zone de 2K pour les blocs à transférer. Ce buffer est placé dans la limite supérieure de la mémoire. Si cette zone est toutefois occupée par des variables, ces variables sont recopiées dans une autre zone de la mémoire. Ce procédé est comparable à la redoutable garbage collection qui se produit toujours lorsqu'il n'y a plus de place suffisante en mémoire pour les chaînes de caractères et les tableaux.

Le délai d'attente provoqué par le transfert des variables peut cependant être notablement réduit si ce buffer de 2K est déjà installé et protégé au début de chaque programme. Un début de programme possible pourrait se présenter ainsi:

```
10 OPENOUT "DUMMY"
20 MEMORY HIMEM - 1
30 CLOSEOUT
40
50 'RESTE DU PROGRAMME
```

Ce procédé n'a bien sûr de sens que si vous travaillez dans le programme en question avec des fichiers. Si ce n'est pas le cas, vous pouvez renoncer à ces lignes de programme et entrer simplement l'instruction CLEAR avant la sauvegarde. Toutes les variables définies auparavant seront ainsi supprimées et l'installation du buffer de cassette se fera sans délai notable.

1.10.5 L'interface d'imprimante centronics

On trouve sur tout ordinateur quelque chose qu'on considère comme pouvant être amélioré. Sur le CPC, c'est sans conteste l'interface imprimante. Bien que de nombreux points faibles ou défauts du CPC 464 aient été éliminés sur ses successeurs 664 et 6128, aucune modification n'a été apportée au point faible le plus regrettable, l'interface imprimante. La cause de notre 'mauvaise humeur' est le fait que l'interface ne dispose toujours que de 7 bits. La plupart des imprimantes, y compris celle proposée par AMSTRAD pour le CPC, ont une entrée 8 bits et donc de nombreuses commandes et possibilités de ces imprimantes ne peuvent être obtenues que par des détours, ou même ne peuvent pas être obtenues du tout.

Mais considérons d'abord la structure électronique de cette interface.
L'interface se compose principalement d'un octuple latch 74LS273. Les huit différents latchs travaillent comme des flip-flops, l'information envoyée sur les entrées est stockée avec une bascule high-low sur l'entrée d'horloge pin 11 et elle est disponible sur les sorties, jusqu'à un RESET ou à une nouvelle programmation, quelles que soient les modifications sur les signaux d'entrée.

Le signal d'horloge dont la bascule high-low déclenche le stockage des valeurs d'entrée est produit avec une porte logique OR. La sortie pin 11 devient low, lorsque les deux entrées sont low.
La connexion de l'imprimante est également appelée à travers l'adressage de port. C'est pourquoi le signal IOWR* se trouve sur une entrée de la porte logique OR et que le canal d'adresse A12 se trouve sur l'autre entrée.

Comme sur les autres éléments périphériques, le décodage est ici donc également très incomplet. Les canaux d'adresse qui ne sont pas utilisés pour le décodage doivent donc être high pour éviter des collisions avec d'autres adresses de port utilisées. Ceci donne une adresse de port effective de &EFxx.

Les entrées du latch de l'imprimante sont reliées au bus de données du processeur. Les sorties se trouvent sur la connexion de l'imprimante. Seul le bit 7 est envoyé au port Centronics à travers une porte logique NAND utilisée comme inverseur. Ce bit représente le signal strobe nécessité par l'imprimante. Ce signal est normalement high. Mais si l'ordinateur veut envoyer un caractère à l'imprimante, il envoie l'octet à transmettre sur les canaux de données et place peu après le signal strobe sur low. L'octet à transmettre est ainsi accepté par l'imprimante.
A condition toutefois que le signal busy de l'imprimante soit low. L'état du signal busy est interrogé par le bit 6 du port B du 8255.

Mais comment le signal strobe peut-il être produit? Rien de plus simple.
Chaque octet à transmettre est d'abord ANDé avec &7F. Le bit supérieur de l'octet est ainsi supprimé de façon certaine. Cet octet est sorti sur le port de l'imprimante par une instruction OUT.
Les bits à transmettre se trouvent maintenant déjà sur l'imprimante, mais le signal strobe est toujours high, à travers l'inverseur. C'est pourquoi on met ensuite avec OR &80 le bit 7 de la valeur à sortir qui est également sortie sur le port imprimante. La valeur à transmettre n'a pas été modifiée, seul le signal strobe est devenu low à travers l'inverseur.
Ce signal doit cependant redevenir également high, c'est pourquoi le bit supérieur est à nouveau supprimé avec AND et l'octet est à nouveau sorti. Un octet a été ainsi envoyé de l'ordinateur à l'imprimante.

La sortie sur l'imprimante ne pose pas de problème en Basic. Mais même en langage-machine, il n'est pas nécessaire d'écrire soi-même toute cette procédure. Il y a plusieurs routines système qui vous évitent une bonne part de ce travail de programmation.
Il y a d'abord la routine dont l'entrée est en &BD2B. A travers cette routine, vous pouvez sortir un caractère sur l'imprimante. Le caractère doit chaque fois se trouver dans l'accumulateur. Cette routine teste en outre si l'imprimante est 'busy'. Si l'imprimante ne répond pas dans un délai de 0.4 secondes, la routine revient

avec un flag carry nul.
Il faut alors faire une nouvelle tentative avec le même caractère. Cette routine est également utilisée par l'interpréteur Basic. Si la transmission est réussie, le carry est mis. Le prochain caractère peut alors être envoyé.

Une autre routine a son entrée trois octets plus loin (&BD2E). Cette routine peut être utilisée pour examiner l'état de l'imprimante. Si aucune imprimante n'est connectée ou si l'imprimante répond 'busy', si elle ne peut donc pas recevoir de caractères pour le moment, cette routine revient avec un carry mis, sinon le carry est annulé.

La troisième routine exploitable (&BD31) accomplit toutes les procédures nécessaires à la sortie d'un caractère sur l'imprimante. Le programmeur doit cependant tester alors auparavant si l'imprimante est prête à recevoir puis transmettre le caractère voulu dans l'accumulateur. Si le test de l'état de l'imprimante est négligé, le caractère peut éventuellement se perdre dans le 'vide'.

Comment ces routines peuvent être mises en oeuvre, nous vous l'indiquerons plus tard dans cet ouvrage. Nous vous montrerons en effet pour l'exemple d'un hardcopy de texte et de graphisme, comment utiliser ces routines et d'autres.

Mais il convient de tenir compte d'une autre particularité de cette connexion Centronics.
La disposition des contacts du port d'imprimante incite à se procurer les fiches nécessaires ainsi qu'un bout de câble plat pour réaliser soi-même un tel câble. Si les connecteurs sont en outre des pinces crocodile, même des possesseurs de CPC peu doués manuellement peuvent réaliser un tel câble en 5 à 10 minutes. Toutes les imprimantes Centronics peuvent être alors utilisées.
Mais lors du premier essai de fonctionnement, vous aurez une grosse surprise. L'imprimante dépense curieusement le papier très généreusement. Une ligne vide est ajoutée après chaque ligne imprimée.

La raison en est la suivante:

Le CPC ajoute à la fin de chaque ligne imprimée la suite de caractères CR/LF (Carriage Return, Line Feed) c'est-à-dire la suite d'instructions pour retour de chariot et passage à la ligne. Le papier avance donc d'une ligne. De plus, et sans raison très claire, le pin 14 de la connexion centronics du CPC est cependant encore relié à la masse. Cela produit sur la plus part des imprimantes un passage à la ligne supplémentaire, de sorte qu'une ligne vide est ainsi toujours produite.

La solution est dans ce cas l'interruption du canal menant au pin 14. Après avoir écarté ce canal et éventuellement installé des commutateurs dans l'imprimante si nécessaire comme par exemple sur Epson, tout devrait fonctionner correctement.

1.10.6 La connexion du joystick

La connexion du joystick est certainement utilisée principalement dans un but qui justifie son nom: comme entrée pour l'interrogation d'un joystick. A travers 7 des 9 connexions disponibles, il est cependant également possible d'interroger d'autres touches ou commutateurs. Par programmation et en renonçant aux interruptions et à l'interrogation du clavier, ces sept connexions pourraient même être employées comme sortie. Les connexions de joystick sont en effet reliées au port bi-directionnel du chip sonore et pourraient travailler comme sortie, sous les contraintes indiquées. Le port Centronics est cependant plus facile à manipuler pour effectuer une sortie.

Comme nous l'avons déjà décrit au chapitre 1.10.1, les joysticks sont considérés comme des touches du clavier. C'est pour cette raison que les 7 entrées nécessaires du port du chip sonore sont placées sur la prise du joystick. Deux sorties du décodeur BCD-décimal évoqué lors de la description du clavier sont encore en outre placées sur la prise.

Tous les cinquantièmes de seconde, le clavier est interrogé entièrement. L'état des joysticks est également interrogé à cette occasion. Pour les programmeurs Basic, l'état des joysticks est fourni par la fonction JOY(numéro). L'état des joysticks pourrait être également déterminé simplement avec INKEY. Mais également pour les fanas de l'assembleur, il est possible de déterminer facilement l'état des joysticks. La routine système &BB24 fournit dans le registre double HL l'état actuel des joysticks. En appelant cette routine, on obtient l'état du joystick 0 dans le registre H et le registre L vaut pour le joystick 1. Le codage des touches joystick suit le même schéma qu'avec la fonction JOY(x).

1.10.7 Le connecteur d'extension

Cette interface est la plus universelle du CPC. Sur cette carte de conducteurs à 50 pôles se trouvent, outre tous les signaux du processeur, différents signaux de commande. C'est ici que sont connectées toutes les extensions du système.

La signification des signaux 3 à 39 nous est connue puisqu'elle découle de la description du processeur. C'est pourquoi nous allons nous limiter ici aux connexions restantes.

Sur le pin 1 figure encore une fois le signal sonore. Ce signal n'est toutefois que mono, les trois canaux sont conduits ici.

Les pin 2 et 49 sont reliés à la masse de l'alimentation électrique.

Une particularité est constituée par le signal BUS-RESET* sur le pin 40. En plaçant ce signal à low, on provoque un reset du système.
Malheureusement, le CPC vide toute la mémoire lors d'un reset. Ce signal n'est donc comme signal d'alarme pas plus efficace que le fait de couper puis de rallumer l'ordinateur.

Sur le pin 41 figure le signal reset proprement dit pour les extensions extérieures. Notez cependant que tous les composants ne

peuvent pas être alimentés avec ce signal.
Le 8255 a par exemple besoin de ce signal sous sa forme inversée.

Les deux signaux ROMEN* et ROMDIS sont très intéressants. Le signal ROMEN* qui se trouve sur le pin 42 signale par son niveau low un accès à la Rom intégrée de 32 K. Cet accès peut cependant être interdit par un niveau high sur le pin 43, ROMDIS. La totalité de la Rom intégrée peut donc être ainsi remplacée par des Roms ou Eproms extérieures.
Par un décodage approprié des canaux d'adresse, il est cependant également possible de ne masquer et remplacer que des zones déterminées de la Rom intégrée.

Les deux signaux RAMRD* et RAMDIS ont une fonction semblable pour les accès en lecture sur la Ram interne. Ces signaux sur les pins 43 et 44 peuvent être utilisés pour échanger par exemple des zones de mémoire déterminées avec des Roms ou même des Rams.
La commande de Rams extérieures n'est cependant pas très simple sur le CPC. La principale difficulté vient du fait que le signal WR* pour les Rams internes n'est pas produit par le processeur mais par le Gate Array. Cette impulsion d'écriture ne peut malheureusement (à notre connaissance) être empêchée par aucune astuce de programmation, de sorte qu'un accès en écriture à une Ram externe adresse toujours également et écrit sur la Ram interne.

Le signal CURSOR envoyé sur le pin 46 est fourni avec une programmation appropriée par le contrôleur vidéo. Le CRTC dispose en effet de la possibilité offerte par le curseur électronique. Suivant la programmation, un signal carré d'une fréquence d'environ 1.5 ou 3 Hertz apparaît sur cette sortie. Mais il est également possible de programmer sur cette connexion des niveaux low ou high permanents.
Après l'allumage du CPC, c'est un niveau low permanent qui figure ici.

L'entrée LPEN (Light Pen) sur le pin 47 est reliée directement avec l'entrée light-pen du CRTC. Ce circuit intégré dispose de tous les registres nécessaires pour la gestion du light pen.

L'utilisation du light pen, surtout en graphisme haute résolution est cependant difficilement réalisable sur le CPC car le contrôleur vidéo fournit certes l'adresse MA de la position actuelle du light-pen mais il n'indique pas l'adresse RA actuelle. Du fait de la structure spéciale de la Ram vidéo, cette indication est cependant nécessaire si l'on veut dessiner sur l'écran avec le light-pen.

L'entrée pin 48 porte la désignation EXP* et est reliée au port B du 8255 Bit 4. Une extension extérieure peut placer cette connexion sur la masse et se faire ainsi remarquer par le système d'exploitation.

Le dernier signal à évoquer, sur le Pin 50, est le signal d'horloge du processeur. Ce signal, avec une fréquence de 4 MHz, est par exemple utilisé par le contrôleur du lecteur de disquette.

2 LE SYSTEME D'EXPLOITATION

Derrière ce nom qui ne dit rien au non-initié, se cache le coeur de l'ordinateur. C'est ici qu'est réalisée la liaison entre programme de l'utilisateur et le matériel.
L'intérpréteur Basic doit à cet égard être considéré lui-même comme un programme qui accède à travers le système d'exploitation à l'électronique de l'ordinateur.

La structure du système d'exploitation est organisée logiquement et clairement en sections ou packs dont chacune a une fonction particulière. Cela commence au niveau inférieur par le MACHINE PACK qui est la partie la plus proche de l'électronique et qui sert par exemple le port d'imprimante, les registres de son, etc..., cela continue avec le SCREEN PACK qui contrôle l'écran et qui est appelé par le TEXT PACK ou le GRAPHICS PACK.

Un examen plus approfondi montre que chaque pack est strictement délimité et fermé et que la communication avec les autres packs ne se fait qu'à travers certaines interfaces bien définies. En outre, chaque pack dispose d'une zone de Ram propre qu'il emploie comme mémoire de travail. L'appel des routines se produit en règle générale à travers des vecteurs de la Ram ou, plus rarement, directement à travers l'adresse de la Rom.

Cela incline à supposer que le système d'exploitation, probablement à cause de peu de temps disponible, a été écrit par plusieurs programmeurs, chacun étant responsable d'un ou plusieurs packs et après qu'on se soit entendu uniquement sur les interfaces.

Quoi qu'il en soit, cette structure claire et l'accès par des vecteurs à tous les coins et recoins ouvrent au programmeur des horizons insoupçonnés et tout à fait inconnus jusqu'ici.

Citons simplement comme exemple la possibilité d'écrire une routine pour une véritable imprimante 8 bits (sans parler du problème de la connexion) et de rendre cette routine utilisable par le système simplement en modifiant le vecteur MC WAIT PRINTER.

Cette indication doit également vous servir d'avertissement: ne craignez pas d'utiliser les routines du système d'exploitation, mais ne les utilisez qu'à travers les vecteurs! Il se pourrait en effet que quelqu'un d'autre (cartouche Rom) ait déplacé quelques vecteurs pour faire exécuter certaines fonctions par des routines propres.

Vous constaterez à l'usage qu'il est possible d'écrire des programmes propres en un minimum de temps, pour peu qu'on utilise scrupuleusement les vecteurs. Ce qui est entièrement nouveau, c'est que même les routines arithmétiques du Basic tournent avec ce mécanisme ce qui peut vous permettre d'une part d'y faire exécuter vos propres calculs et d'autre part d'y placer vos propres programmes si vous souhaitez par exemple une plus grande précision.

Puisque nous vous avons montré notre enthousiasme pour les vecteurs, c'est aussi avec eux que nous commencerons dans le chapitre suivant.

2.1 Les vecteurs du système d'exploitation

Nous vous présentons dans les pages suivantes les adresses de la RAM à travers lesquelles vous pouvez appeler des routines du systèmes d'exploitation ou que vous pouvez modifier de façon à faire exécuter certaines fonctions par vos propres programmes. Il s'agit en partie de routines complètes qui ont été copiées dans la RAM et au milieu desquelles vous pouvez sauter et en partie aussi de RST 1 ou RST 5 suivi de l'adresse INLINE qui renvoie à la ROM.

Vous trouverez en annexe une liste des routines de la ROM qui vous aidera à retrouver rapidement l'ensemble d'une routine que vous recherchez.

2.1.1 Les vecteurs du système d'exploitation du CPC 664

B900	KL U ROM ENABLE connecter ROM supérieure actuelle.
B903	KL U ROM DISABLE déconnecter ROM supérieure.
B906	KL L ROM ENABLE connecter ROM inférieure.
B909	KL L ROM DISABLE déconnecter ROM inférieure.
B90C	KL ROM RESTORE restaurer ancienne configuration ROM
B90F	KL ROM SELECT sélectionner une ROM supérieure déterminée.
B912	KL CURR SELECTION quelle ROM supérieure est activée?
B915	KL PROBE ROM examiner ROM
B918	KL ROM DESELECT restaurer ancienne configuration ROM supérieure.
B91B	KL LDIR LDIR pour ROMs bloquées.
B91E	KL LDDR LDDR pour ROMs bloquées.
B921	KL POLL SYNCHRONOUS Y a-t-il un event de priorité supérieure à celle de l'actuel?
B941	RST 7 INTERRUPT ENTRY CONT'D entrée pour interruptions hardware.
B978	KL EXT INTERRUPT ENTRY
B984	KL LOW PCHL CONT'D saut à la ROM ou RAM inférieures.

B98A RST 1 LOW JUMP CONT'D Appel d'une routine dans le système d'exploitation ou dans la RAM lui étant parallèle.

B9B9 KL FAR PCHL CONT'D

B9C1 KL FAR ICALL CONT'D

B9C7 RST 3 LOW FAR CALL CONT'D On peut appeler une routine n'importe où en RAM ou en ROM.

BA17 KL SIDE PCHL CONT'D

BA1D RST 2 LOW SIDE CALL CONT'D Sert à appeler une routine dans la ROM d'extension.

BA35 RST 5 FIRM JUMP CONT'D permet de sauter à une routine du système d'exploitation.

BA51 KL L ROM ENABLE CONT'D connecter ROM inférieure.

BA58 KL L ROM DISABLE CONT'D déconnecter ROM inférieure.

BA5F KL U ROM ENABLE CONT'D activer ROM supérieure.

BA66 KL U ROM DISABLE CONT'D désactiver ROM supérieure.

BA70 KL ROM RESTORE CONT'D restaurer ancienne configuration ROM.

BA79 KL ROM SELECT CONT'D sélectionner une ROM supérieure déterminée.

BA7E KL PROBE ROM CONT'D examiner ROM.

BA87 KL ROM DESELECT CONT'D restaurer ancienne configuration ROM supérieure.

BA9D KL CURR SELECTION CONT'D quelle ROM supérieure est activée?

BAA1 KL LDIR CONT'D LDIR pour ROMs bloquées.

BAA7 KL LDDR CONT'D LDDR pour ROMs bloquées.

BAAD KL ROM OFF & CONFIG. SAVE

BAC6 RST 4 RAM LAM CONT'D lire contenu RAM, indépendamment de l'état ROM

BAD7 KL RAM LAM (IX) correspond à ld a,(ix).

BB00 KM INITIALISE initialisation complète de la gestion clavier.

BB03 KM RESET réinitialisation de la gestion clavier.

BB06 KM WAIT CHAR attendre un caractère du clavier.

BB09 KM READ CHAR aller chercher un caractère au clavier s'il y a un caractère.

BB0C KM CHAR RETURN placer caractère dans le buffer clavier pour le prochain accès.

BB0F KM SET EXPAND créer chaîne d'extension.

BB12 KM GET EXPAND aller chercher caractère dans chaîne d'extension.

BB15 KM EXP BUFFER Affecter mémoire pour chaîne d'extension.

BB18 KM WAIT KEY attendre la frappe d'une touche.

BB1B KM READ KEY aller chercher numéro de touche si une touche a été enfoncée.

BB1E KM TEST KEY une touche a-t-elle été enfoncée?

BB21 KM GET STATE aller chercher état SHIFT.

BB24 KM GET JOYSTICK Interrogation de l'état actuel du joystick.

BB27 KM SET TRANSLATE effectuer une entrée dans la table clavier (1. niveau).

BB2A KM GET TRANSLATE aller chercher une entrée de la table clavier (1. niveau).

BB2D KM SET SHIFT effectuer une entrée dans la table clavier (2. niveau).

BB30 KM GET SHIFT aller chercher une entrée de la table clavier (2. niveau).

BB33 KM SET CONTROL effectuer une entrée dans la table clavier (3. niveau).

BB36 KM GET CONTROL aller chercher une entrée de la table clavier (3. niveau).

BB39 KM SET REPEAT fixer fonction de répétition pour une touche déterminée.

BB3C KM GET REPEAT fonction de répétition fixée pour une touche déterminée?

BB3F KM SET DELAY fixer emploi et vitesse de répétition de touche.

BB42 KM GET DELAY aller chercher paramètres pour emploi et vitesse de la répétition de touches.

BB45 KM ARM BREAK autoriser la touche Break.

BB48 KM DISARM BREAK verrouiller la touche Break.

BB4B KM BREAK EVENT exécuter routines lorsque la touche Break est appuyée.

BB4E TXT INITIALISE initialisation complète du pack TEXTE.

BB51 TXT RESET réinitialisation du pack TEXTE.

BB54 TXT VDU ENABLE On peut écrire des caractères sur l'écran.

BB57 TXT VDU DISABLE inhiber représentation du caractère.

BB5A TXT OUTPUT représenter caractère (de commande) ou l'exécuter.

BB5D TXT WR CHAR représenter caractère.

BB60 TXT RD CHAR lire un caractère de l'écran.

BB63 TXT SET GRAPHIC activer ou désactiver représentation des caractères de commande.

BB66 TXT WIN ENABLE déterminer taille de la fenêtre de texte actuelle.

BB69 TXT GET WINDOW quelle taille a la fenêtre de texte actuelle?

BB6C TXT CLEAR WINDOW vider fenêtre de texte actuelle.

BB6F TXT SET COLUMN fixer position horizontale du curseur.

BB72 TXT SET ROW fixer position verticale du curseur.

BB75 TXT SET CURSOR positionner le curseur.

BB78 TXT GET CURSOR demander la position actuelle du curseur.

BB7B TXT CUR ENABLE autoriser curseur (programme utilisateur).

BB7E TXT CUR DISABLE verrouiller curseur (programme utilisateur).

BB81 TXT CUR ON autoriser curseur (système d'exploitation).

BB84 TXT CUR OFF verrouiller curseur (système d'exploitation, priorité supérieure à BB7B TXT CUR ENABLE/BB7E TXT CUR DISABLE).

BB87 TXT VALIDATE curseur à l'intérieur de la fenêtre de texte?

BB8A TXT PLACE/REMOVE CURSOR fixer curseur sur l'écran/enlever curseur de l'écran.

BB8D TXT PLACE/REMOVE CURSOR fixer curseur sur l'écran/enlever curseur de l'écran.

BB90 TXT SET PEN fixer couleur de premier plan.

BB93 TXT GET PEN quelle couleur de premier plan?

BB96 TXT SET PAPER fixer couleur d'arrière-plan.

BB99 TXT GET PAPER quelle couleur de fond?

BB9C TXT INVERSE échanger couleurs de premier et arrière plans actuelles.

BB9F TXT SET BACK mode transparent activé/désactivé.

BBA2 TXT GET BACK quel mode transparent?

BBA5 TXT GET MATRIX aller chercher adresse du modèle points d'un caractère.

BBA8 TXT SET MATRIX fixer adresse du modèle points (défini par l'utilisateur) d'un caractère déterminé.

BBAB TXT SET M TABLE fixer adresse de départ et premier caractère d'une matrice de points définie par l'utilisateur

BBAE TXT GET M TABLE adresse de départ et premier caractère d'une matrice utilisateur?

BBB1 TXT GET CONTROLS aller chercher adresse de la table de saut des caractères de commande.

BBB4 TXT STR SELECT sélectionner fenêtre de texte.

BBB7 TXT SWAP STREAMS échange des paramètres (couleurs, limites de fenêtre etc.) de deux fenêtres de texte.

BBBA GRA INITIALISE initialisation complète du pack graphique.

BBBD GRA RESET réinitialisation du pack graphique.

BBC0 GRA MOVE ABSOLUTE déplacement vers une position absolue.

BBC3 GRA MOVE RELATIVE Déplacement relativement à la position actuelle.

BBC6 GRA ASK CURSOR Où est le curseur graphique actuel?

BBC9 GRA SET ORIGIN fixer origine des coordonnées utilisateur.

BBCC GRA GET ORIGIN aller chercher origine des coordonnées utilisateur.

BBCF GRA WIN WIDTH fixer limites gauche et droite de la fenêtre graphique.

BBD2 GRA WIN HEIGHT fixer limites supérieure et inférieure de la fenêtre graphique.

BBD5 GRA GET W WIDTH limites gauche et droite de la fenêtre graphique?

BBD8 GRA GET W HEIGHT limites supérieure et inférieure de la fenêtre graphique?

BBDB GRA CLEAR WINDOW vider fenêtre graphique.

BBDE GRA SET PEN fixer couleur d'écriture.

BBE1 GRA GET PEN quelle couleur d'écriture?

BBE4 GRA SET PAPER fixer couleur d'arrière-plan.

BBE7 GRA GET PAPER quelle couleur de fond?

BBEA GRA PLOT ABSOLUTE fixer un point graphique (absolu).

BBED GRA PLOT RELATIVE fixer point graphique (relativement au curseur actuel).

BBF0 GRA TEST ABSOLUTE point fixé (absolu)?

BBF3 GRA TEST RELATIVE point fixé (relativement au curseur actuel)?

BBF6 GRA LINE ABSOLUTE tracer une ligne de position act. à position absolue.

BBF9 GRA LINE RELATIVE tracer ligne de distance act. à distance rel.

BBFC GRA WR CHAR écrire un caractère dans la position curseur graphique actuelle.

BBFF SCR INITIALISE initialisation du pack écran.

BC02 SCR RESET réinitialisation du pack écran.

BC05 SCR SET OFFSET fixer adresse de départ du premier caractère relativement à l'adresse de base de la RAM vidéo

BC08 SCR SET BASE fixer adresse de base de la RAM vidéo.

BC0B SCR GET LOCATION act. début écran? (base+Offset).

BC0E SCR SET MODE fixer mode écran.

BC11 SCR GET MODE aller chercher mode écran et tester

BC14 SCR CLEAR vider l'écran.

BC17 SCR CHAR LIMITS aller chercher nombres maxi de lignes et colonnes de l'écran (dépend du mode).

BC1A SCR CHAR POSITION convertir coordonnées physiques en position écran.

BC1D SCR DOT POSITION déterminer position écran pour un pixel.

BC20 SCR NEXT BYTE augmenter une adresse écran fournie d'une position de caractère.

BC23 SCR PREV BYTE diminuer une adresse écran fournie d'une position

BC26 SCR NEXT LINE augmenter une adresse écran d'une ligne.

BC29 SCR PREV LINE diminuer une adresse écran d'une ligne.

BC2C SCR INK ENCODE transposer une ink en format codé.

BC2F SCR INK DECODE décoder une ink.

BC32 SCR SET INK affecter couleur(s) à un No ink.

BC35 SCR GET INK couleur(s) d'un No ink?

BC38 SCR SET BORDER fixer couleur(s) du bord.

BC3B SCR GET BORDER couleur(s) du bord?

BC3E SCR SET FLASHING fixer durées de clignotement.

BC41 SCR GET FLASHING durées de clignotement?

BC41 SCR GET FLASHING durées de clignotement?
remplir fenêtre indiquée avec une couleur (les positions sont des adresses écran indépendantes du mode)

BC4A SCR CHAR INVERT échanger pour une caractère les couleurs de premier et d'arrière plans.

BC4A SCR CHAR INVERT échanger pour une caractère les couleurs de premier et d'arrière plans.

BC50 SCR SW ROLL écran une ligne vers le haut ou le bas (software).

BC53 SCR UNPACK agrandir matrice caractère (pour modes 0/1)

BC56 SCR REPACK ramener la matrice caractère à sa forme originelle.

BC59 SCR ACCESS fixer caractères de commande visibles/ invisibles.

BC5C SCR PIXELS fixer points sur l'écran.

BC5F SCR HORIZONTAL tracer ligne horizontale.

BC62 SCR VERTICAL tracer ligne verticale.

BC65 CAS INITIALISE initialiser manager cassette.

BC68 CAS SET SPEED fixer vitesse d'écriture.

BC6B CAS NOISY message cassette activé/désactivé.

BC6E CAS START MOTOR lancer moteur cassette.

BC71 CAS STOP MOTOR arrêter moteur cassette.

BC74 CAS RESTORE MOTOR restaurer ancien état moteur.

BC77 CAS IN OPEN ouverture d'un fichier d'entrée.

BC7A CAS IN CLOSE fermeture correcte d'un fichier d'entrée.

BC7D CAS IN ABANDON fermer immédiatement fichier d'entrée.

BC80 CAS IN CHAR lire caractère (du buffer).

BC83 CAS IN DIRECT amener fichier entier dans la mémoire.

BC86 CAS RETURN renvoyer dernier caractère lu dans le buffer.

BC89 CAS TEST EOF fin de fichier?

BC8C CAS OUT OPEN ouverture correcte d'un fichier de sortie.

BC8F CAS OUT CLOSE fermeture correcte d'un fichier de sortie.

BC92 CAS OUT ABANDON fermer immédiatement fichier de sortie.

BC95 CAS OUT CHAR écrire caractère (dans le buffer).

BC98 CAS OUT DIRECT écrire zone mémoire définie sur cassette (pas à travers le buffer).

BC9B CAS CATALOG Sort un catalogue de la cassette sur l'écran.

BC9E CAS WRITE écrire bloc.

BCA1   CAS READ lire bloc.

BCA4   CAS CHECK comparer bloc sur la bande avec contenu de la mémoire.

BCA7   SOUND RESET réinitialisation du pack Sound.

BCAA   SOUND QUEUE ajouter note à la file d'attente.

BCAD   SOUND CHECK Y a-t-il encore de la place dans la file d'attente?

BCB0   SOUND ARM EVENT 'armer' bloc event pour le cas où une place se libèrerait dans la file d'attente.

BCB3   SOUND RELEASE autoriser notes.

BCB6   SOUND HOLD arrêter notes immédiatement

BCB9   SOUND CONTINUE poursuivre le traitement des notes arrêtées auparavant.

BCBC   SOUND AMPL ENVELOPE créer courbe d'enveloppe de volume.

BCBF   SOUND TONE ENVELOPE créer courbe d'enveloppe de note.

BCC2   SOUND A ADRESS aller chercher adresse d'une courbe d'enveloppe.

BCC5   SOUND T ADRESS aller chercher adresse d'une courbe d'enveloppe de note

BCC8   KL CHOKE OFF réinitialiser le Kernal

BCCB   KL ROM WALK extensions ROM quelconques?

BCCE   KL INIT BACK ajouter extension ROM.

BCD1   KL LOG EXT ajouter extension résidente.

BCD4   KL FIND COMMAND chercher instruction dans toutes les zones mémoire ajoutées.

BCD7   KL NEW FRAME FLY créer et ajouter bloc event.

BCDA   KL ADD FRAME FLY ajouter bloc event.

BCDD   KL DEL FRAME FLY supprimer bloc event.

BCE0   KL NEW FAST TICKER comme BCD7.

BCE3   KL ADD FAST TICKER comme BCDA.

BCE6   KL DEL FAST TICKER comme BCDD.

BCE9   KL ADD TICKER ajouter bloc ticker.

BCEC   KL DEL TICKER supprimer bloc ticker.

BCEF   KL INIT EVENT créer bloc event.

BCF2   KL EVENT 'kick' bloc event.

BCF5   KL SYNC RESET supprimer Sync Pending Queue.

BCF8 KL DEL SYNCHRONOUS supprimer un bloc déterminé de la pending queue.

BCFB KL NEXT SYNC Au suivant.

BCFE KL DO SYNC exécuter routine event.

BD01 KL DONE SYNC routine event terminé.

BD04 KL EVENT DISABLE Verrouillage des évènements normalement simultanés. Les évènements urgents simultanés ne sont pas verrouillés.

BD07 KL EVENT ENABLE autoriser évènements simultanés normaux.

BD0A KL DISARM EVENT verrouiller bloc event (compteur negatif).

BD0D KL TIME PLEASE Combien de temps s'est-il écoulé?

BD10 KL TIME SET Fixer le temps sur valeur indiquée.

BD13 MC BOOT PROGRAM restaure le système d'exploitation et transmet la commande à une routine en (hl).

BD16 MC START PROGRAM initialisation du système et appel d'un programme

BD19 MC WAIT FLYBACK attendre retour du faisceau.

BD1C MC SET MODE fixer mode écran.

BD1F MC SCREEN OFFSET fixer offset écran.

BD22 MC CLEAR INKS fixer bord écran et inks sur une couleur.

BD25 MC SET INKS fixer couleur pour toutes les inks.

BD28 MC RESET PRINTER réinitialisation du point de branchement indirect pour l'imprimante.

BD2B MC PRINT CHAR imprimer caractère si possible.

BD2E MC BUSY PRINTER imprimante encore occupée?

BD31 MC SEND PRINTER imprimer caractère (attendre jusqu'à ce que ce soit possible).

BD34 MC SOUND REGISTER fournir des données au Sound Controller

BD37 JUMP RESTORE initialiser tous les vecteurs de saut.

BD3A KM SET STATE

BD3D KM VIDER BUFFER

BD40 TXT FLAG CURSEUR ACTUEL VERS ACCU

BD43 GRA NN

BD46 GRA SAUVER PARAMETRES

BD49 GRA SAUVER PARAMETRES MASQUE

BD4C GRA SAUVER PARAMETRES MASQUE

BD4F GRA CONVERTIR COORD. coordonnées logiques en coordonnées physiques.

BD52 GRA FILL Fillroutine

BD55 SCR MODIFIER DEBUT ECRAN

BD58 MC AFFECTATION DE CARACTERES

Les vecteurs suivants sont utilisés par le Basic.

BD5B EDIT

BD5E FLO COPIER VARIABLE DE (DE) VERS (HL)

BD61 FLO ENTIER VERS VIRGULE FLOTTANTE

BD64 FLO VALEUR 4 OCTETS VERS FLO

BD67 FLO FLO VERS ENTIER

BD6A FLO FLO VERS ENTIER

BD6D FLO FIX

BD70 FLO INT

BD73 FLO

BD79 FLO ADDITION

BD79 FLO ADDITION

BD7C FLO RND

BD7F FLO SOUSTRACTION

BD82 FLO MULTIPLICATION

BD85 FLO DIVISION

BD88 FLO ALLER CHERCHER DERNIERE VALEUR RND

BD8B FLO COMPARAISON

BD8E FLO CHANGEMENT DE SIGNE

BD91 FLO SGN

BD94 FLO DEG/RAD

BD97 FLO PI

BD9A FLO SQR

BD9D FLO ELEVATION A LA PUISSANCE

BDA0 FLO LOG

BDA3 FLO LOG10

BDA6 FLO EXP

BDA9 FLO SIN
BDAC FLO COS
BDAF FLO TAN
BDB2 FLO ATN
BDB5 FLO VALEUR 4 OCTETS VERS FLO
BDB8 FLO RND INIT
BDBB FLO SET RND SEED

Ici commencent ce qu'on appelle les INDIRECTIONS. Il s'agit de sauts dans le système d'exploitation qui ne sont pas traités globalement mais individuellement par chaque pack lorsque son RESET ou INITIALISE est parcouru.

BDCD TXT DRAW/UNDRAW CURSOR fixation/suppression du curseur.
BDD0 TXT DRAW/UNDRAW CURSOR fixation/suppression du curseur
BDD3 TXT WRITE CHAR écrire un caractère sur l'écran.
BDD6 TXT UNWRITE CHAR lire un caractère de l'écran.
BDD9 TXT OUT ACTION sortie d'un caractère sur l'écran ou exécution d'un code de commande.

BDDC GRA PLOT représenter un point sur l'écran.
BDDF GRA TEST fournit l'ink de la position graphique actuelle.
BDE2 GRA LINE dessin d'une ligne.

BDE5 SCR READ lecture d'un pixel et décodage d'une ink.
BDE8 SCR WRITE écrire pixel(s).
BDEB SCR CLEAR vidage de l'écran.

BDEE KM TEST BREAK ESC, SHIFT et CTRL entraînent une réinitialisation totale du système.

BDF1 MC WAIT PRINTER envoyer un caractère à l'imprimante; si celle-ci n'est pas prête, attendre une période de délai.

BDF4 KM UPDATE KEY STATE MAP

2.1.2 Les vecteurs du système d'exploitation du CPC 6128

B900 KL U ROM ENABLE connecter ROM supérieure actuelle.

B903 KL U ROM DISABLE déconnecter ROM supérieure.

B906 KL L ROM ENABLE connecter ROM inférieure.

B909 KL L ROM DISABLE déconnecter ROM inférieure.

B90C KL ROM RESTORE restaurer ancienne configuration ROM.

B90F KL ROM SELECT sélectionner une ROM supérieure déterminée.

B912 KL CURR SELECTION quelle ROM supérieure est activée?

B915 KL PROBE ROM examiner ROM.

B918 KL ROM DESELECT restaurer ancienne configuration ROM supérieure.

B91B KL LDIR LDIR pour ROMs bloquées.

B91E KL LDDR LDDR pour ROMs bloquées.

B921 KL POLL SYNCHRONOUS Y a-t-il un event de priorité supérieure à celle de l'actuel?

B941 RST 7 INTERRUPT ENTRY CONT'D entrée pour interruptions hardware.

B978 KL EXT INTERRUPT ENTRY

B984 KL LOW PCHL CONT'D saut à la ROM ou RAM inférieures.

B98A RST 1 LOW JUMP CONT'D Appel d'une routine dans le système d'exploitation ou dans la RAM lui étant parallèle.

B9B9 KL FAR PCHL CONT'D

B9C1 KL FAR ICALL CONT'D

B9C7 RST 3 LOW FAR CALL CONT'D On peut appeler une routine n'importe où en RAM ou en ROM.

BA17 KL SIDE PCHL CONT'D

BA1D RST 2 LOW SIDE CALL CONT'D Sert à appeler une routine dans la ROM d'extension.

BA35 RST 5 FIRM JUMP CONT'D permet de sauter à une routine du système d'exploitation.

BA51 KL L ROM ENABLE CONT'D connecter ROM inférieure.

BA58 KL L ROM DISABLE CONT'D déconnecter ROM inférieure.

BA5F KL U ROM ENABLE CONT'D activer ROM supérieure.

BA66 KL U ROM DISABLE CONT'D déconnecter ROM supérieure.

BA70 KL ROM RESTORE CONT'D restaurer ancienne configuration ROM.

BA79 KL ROM SELECT CONT'D sélectionner une ROM supérieure déterminée.

BA7E KL PROBE ROM CONT'D examiner ROM.

BA87 KL ROM DESELECT CONT'D restaurer ancienne configuration ROM supérieure.

BA9D KL CURR SELECTION CONT'D quelle ROM supérieure est activée?

BAA1 KL LDIR CONT'D LDIR pour ROMs bloquées.

BAA7 KL LDDR CONT'D LDDR pour ROMs bloquées.

BAAD KL ROM OFF & CONFIG. SAVE

BAC6 RST 4 RAM LAM CONT'D lire contenu RAM, indépendamment de l'état ROM.

BB0C KM CHAR RETURN placer caractère dans le buffer clavier pour le prochain accès.

BB00 KM INITIALISE initialisation complète de la gestion clavier.

BB03 KM RESET réinitialisation de la gestion clavier.

BB0C KM CHAR RETURN placer caractère dans le buffer clavier pour le prochain accès.

BB0C KM CHAR RETURN placer caractère dans le buffer clavier pour le prochain accès.

BB0C KM CHAR RETURN placer caractère dans le buffer clavier pour le prochain accès.

BB0F KM SET EXPAND créer chaîne d'extension.

BB12 KM GET EXPAND aller chercher caractère dans chaîne d'extension.

BB15 KM EXP BUFFER Affecter mémoire pour chaîne d'extension.

BB18 KM WAIT KEY attendre la frappe d'une touche.

BB1B KM READ KEY aller chercher numéro de touche si une touche a été enfoncée.

BB1E KM TEST KEY une touche a-t-elle été enfoncée?

BB21 KM GET STATE aller chercher état SHIFT.

BB24 KM GET JOYSTICK Interrogation de l'état actuel du joystick.

BB27 KM SET TRANSLATE effectuer une entrée dans la table clavier (1. niveau).

BB2A KM GET TRANSLATE aller chercher une entrée de la table clavier (1. niveau).

BB2D KM SET SHIFT effectuer une entrée dans la table clavier (2. niveau).

BB30 KM GET SHIFT aller chercher une entrée de la table clavier (2. niveau).

BB33 KM SET CONTROL effectuer une entrée dans la table clavier (3. niveau).

BB36 KM GET CONTROL aller chercher une entrée de la table clavier (3. niveau).

BB39 KM SET REPEAT fixer fonction de répétition pour une touche déterminée.

BB3C KM GET REPEAT fonction de répétition fixée pour une touche déterminée?

BB3F KM SET DELAY fixer emploi et vitesse de répétition de touche.

BB42 KM GET DELAY aller chercher paramètres pour emploi et vitesse de la répétition de touches.

BB45 KM ARM BREAK autoriser la touche BREAK.

BB48 KM DISARM BREAK verrouiller la touche Break.

BB4B KM BREAK EVENT exécuter routines lorsque la touche Break est appuyée.

BB4E TXT INITIALISE initialisation complète du pack TEXTE.

BB51 TXT RESET réinitialisation du pack TEXTE.

BB54 TXT VDU ENABLE On peut écrire des caractères sur l'écran.

BB57 TXT VDU DISABLE inhiber représentation du caractère.

BB5A TXT OUTPUT représenter caractère (de commande) ou l'exécuter.

BB5D TXT WR CHAR représenter caractère.

BB60 TXT RD CHAR lire un caractère de l'écran.

BB63 TXT SET GRAPHIC activer ou désactiver représentation des caractères de commande.

BB66 TXT WIN ENABLE déterminer taille de la fenêtre de texte actuelle.

BB69 TXT GET WINDOW quelle taille a la fenêtre de texte actuelle?

BB6C TXT CLEAR WINDOW supprimer fenêtre de texte actuelle.

BB6F TXT SET COLUMN fixer position horizontale du curseur.

BB72 TXT SET ROW fixer position verticale du curseur.

BB75 TXT SET CURSOR positionner le curseur.

BB78 TXT GET CURSOR demander la position actuelle du curseur.

BB7B TXT CUR ENABLE autoriser curseur (programme utilisateur).

BB7E TXT CUR DISABLE verrouiller curseur (programme utilisateur).

BB81 TXT CUR ON autoriser curseur (système d'exploitation).

BB84 TXT CUR OFF verrouiller curseur (système d'exploitation, priorité supérieure à BB7B TXT CUR ENABLE/BB7E TXT CUR DISABLE).

BB87 TXT VALIDATE curseur à l'intérieur de la fenêtre de texte?

BB8A TXT PLACE/REMOVE CURSOR fixer curseur sur l'écran/enlever curseur de l'écran.

BB8D TXT PLACE/REMOVE CURSOR fixer curseur sur l'écran/enlever curseur de l'écran.

BB90 TXT SET PEN fixer couleur de premier plan.

BB93 TXT GET PEN quelle couleur de premier plan?

BB96 TXT SET PAPER fixer couleur d'arrière-plan.

BB99 TXT GET PAPER quelle couleur de fond?

BB9C TXT INVERSE échanger actuelles couleurs de premier et arrière plans.

BB9F TXT SET BACK mode transparent activé/désactivé.

BBA2 TXT GET BACK quel mode transparent?

BBA5 TXT GET MATRIX aller chercher adresse du modèle points d'un caractère.

BBA8 TXT SET MATRIX fixer adresse du modèle points (défini par l'utilisateur) d'un caractère déterminé.

BBAB TXT SET M TABLE fixer adresse de départ et premier caractère d'une matrice de points définie par l'utilisateur.

BBAE TXT GET M TABLE adresse de départ et premier caractère d'une matrice utilisateur?

BBB1 TXT GET CONTROLS aller chercher adresse de la table de saut des caractères de commande.

BBB4 TXT STR SELECT sélectionner fenêtre de texte.

BBB7 TXT SWAP STREAMS échange des paramètres (couleurs, limites de fenêtre etc.) de deux fenêtres de texte.

BBBA GRA INITIALISE initialisation complète du pack graphique.
BBBD GRA RESET réinitialisation du pack graphique.
BBC0 GRA MOVE ABSOLUTE déplacement vers une position absolue.
BBC3 GRA MOVE RELATIVE Déplacement relativement à la position actuelle.
BBC6 GRA ASK CURSOR Où est le curseur graphique actuel?
BBC9 GRA SET ORIGIN fixer origine des coordonnées utilisateur.
BBCC GRA GET ORIGIN aller chercher origine des coordonnées utilisateur.
BBCF GRA WIN WIDTH fixer limites gauche et droite de la fenêtre graphique.
BBD2 GRA WIN HEIGHT fixer limites supérieure et inférieure de la fenêtre graphique.
BBD5 GRA GET W WIDTH limites gauche et droite de la fenêtre graphique?
BBD8 GRA GET W HEIGHT limites supérieure et inférieure de la fenêtre graphique?
BBDB GRA CLEAR WINDOW vider fenêtre graphique.
BBDE GRA SET PEN fixer couleur d'écriture.
BBE1 GRA GET PEN quelle couleur d'écriture?
BBE4 GRA SET PAPER fixer couleur d'arrière-plan.
BBE7 GRA GET PAPER quelle couleur de fond?
BBEA GRA PLOT ABSOLUTE fixer un point graphique (absolu).
BBED GRA PLOT RELATIVE fixer point graphique (relativement au curseur actuel).
BBF0 GRA TEST ABSOLUTE point fixé (absolu)?
BBF3 GRA TEST RELATIVE point fixé (relativement au curseur actuel)?
BBF6 GRA LINE ABSOLUTE tracer une ligne de position act. à position absolue.
BBF9 GRA LINE RELATIVE tracer ligne de distance act. à distance rel.
BBFC GRA WR CHAR écrire un caractère dans la position curseur graphique actuelle.

BBFF SCR INITIALISE initialisation du pack écran.

BC02 SCR RESET réinitialisation du pack écran.

BC05 SCR SET OFFSET fixer adresse de départ du premier caractère relativement à l'adresse de base de la RAM vidéo.

BC08 SCR SET BASE fixer adresse de base de la RAM vidéo.

BC0B SCR GET LOCATION act. début écran? (base+Offset).

BC0E SCR SET MODE fixer mode écran.

BC11 SCR GET MODE aller chercher mode écran et tester.

BC14 SCR CLEAR vider l'écran.

BC17 SCR CHAR LIMITS aller chercher nombres maxi de lignes et colonnes de l'écran (dépend du mode).

BC1A SCR CHAR POSITION convertir coordonnées physiques en position écran.

BC1D SCR DOT POSITION déterminer position écran pour un pixel.

BC20 SCR NEXT BYTE augmenter une adresse écran fournie d'une position de caractère.

BC23 SCR PREV BYTE diminuer une adresse écran fournie d'une position.

BC26 SCR NEXT LINE augmenter une adresse écran d'une ligne.

BC29 SCR PREV LINE diminuer une adresse écran d'une ligne.

BC2C SCR INK ENCODE transposer une ink en format codé.

BC2F SCR INK DECODE décoder une ink.

BC32 SCR SET INK affecter couleur(s) à un No ink.

BC35 SCR GET INK couleur(s) d'un No ink?

BC38 SCR SET BORDER fixer couleur(s) du bord.

BC3B SCR GET BORDER couleur(s) du bord?

BC3E SCR GET FLASHING durées de clignotement?.

BC41 SCR GET FLASHING durées de clignotement?

BC44 SCR FILL BOX remplir fenêtre indiquée avec une couleur (positions en caractères en fonction du mode)

BC47 SCR FLOOD BOX remplir fenêtre indiquée avec une couleur (les positions sont des adresses écran indépendantes du mode)

BC4A SCR CHAR INVERT échanger couleurs de premier et arrière plans pour un caractère.

BC4D SCR HW ROLL écran d'une ligne vers le haut ou le bas (software).

BC50 SCR SW ROLL écran une ligne vers le haut ou le bas (software).

BC53 SCR UNPACK agrandir matrice caractère (pour modes 0/1).

BC56 SCR REPACK ramener la matrice caractère à sa forme originelle.

BC59 SCR ACCESS fixer caractères de commande visibles/invisibles.

BC5C SCR PIXELS fixer points sur l'écran.

BC5F SCR HORIZONTAL tracer ligne horizontale.

BC62 SCR VERTICAL tracer ligne verticale.

BC65 CAS INITIALISE initialiser manager cassette.

BC68 CAS SET SPEED fixer vitesse d'écriture.

BC6B CAS NOISY message cassette activé/désactivé.

BC6E CAS START MOTOR lancer moteur cassette.

BC71 CAS STOP MOTOR arrêter moteur cassette.

BC74 CAS RESTORE MOTOR restaurer ancien état moteur.

BC77 CAS IN OPEN ouverture d'un fichier d'entrée.

BC7A CAS IN CLOSE fermeture correcte d'un fichier d'entrée.

BC7D CAS IN ABANDON fermer immédiatement fichier d'entrée.

BC80 CAS IN CHAR lire caractère (du buffer).

BC83 CAS IN DIRECT amener fichier entier dans la mémoire.

BC86 CAS RETURN renvoyer dernier caractère lu dans le buffer.

BC89 CAS TEST EOF fin de fichier?

BC8C CAS OUT OPEN ouverture d'un fichier de sortie.

BC8F CAS OUT CLOSE fermeture correcte d'un fichier de sortie.

BC92 CAS OUT ABANDON fermer immédiatement fichier de sortie.

BC95 CAS OUT CHAR écrire caractère (dans le buffer).

BC98 CAS OUT DIRECT écrire zone mémoire définie sur cassette (pas à travers le buffer).

BC9B CAS CATALOG sort un catalogue de la cassette sur l'écran.

BC9E CAS WRITE écrire bloc.

BCA1 CAS READ lire bloc.

BCA4 CAS CHECK comparer bloc sur la bande avec contenu de la mémoire.

BCA7 SOUND RESET réinitialisation du pack sound

BCAA SOUND QUEUE ajouter note à la file d'attente.

BCAD SOUND CHECK encore de la place dans la file d'attente?

BCB0 SOUND ARM EVENT 'armer' bloc event pour le cas où une place se libèrerait dans la file d'attente.

BCB3 SOUND RELEASE autoriser notes.

BCB6 SOUND HOLD arrêter notes immédiatement.

BCB9 SOUND CONTINUE poursuivre le traitement des notes arrêtées auparavant.

BCBC SOUND AMPL ENVELOPE créer courbe d'enveloppe de volume.

BCBF SOUND TONE ENVELOPE créer courbe d'enveloppe de note.

BCC2 SOUND A ADRESS aller chercher adresse d'une courbe d'enveloppe.

BCC5 SOUND T ADRESS aller chercher adresse d'une courbe d'enveloppe de note.

BCC8 KL CHOKE OFF réinitialiser le Kernal.

BCCB KL ROM WALK extensions ROM quelconques?

BCCE KL INIT BACK ajouter extension ROM.

BCD1 KL LOG EXT ajouter extension résidente.

BCD4 KL FIND COMMAND chercher instruction dans toutes les zones mémoire ajoutées.

BCD7 KL NEW FRAME FLY créer et ajouter bloc event.

BCDA KL ADD FRAME FLY ajouter bloc event.

BCDD KL DEL FRAME FLY supprimer bloc event.

BCE0 KL NEW FAST TICKER comme BCD7.

BCE3 KL ADD FAST TICKER comme BCDA.

BCE6 KL DEL FAST TICKER comme BCDD.

BCE9 KL ADD TICKER ajouter bloc ticker.

BCEC KL DEL TICKER supprimer bloc ticker.

BCEF KL INIT EVENT créer bloc event.

BCF2 KL EVENT 'kick' bloc event.

BCF5 KL SYNC RESET supprimer Sync Pending Queue.

BCF8 KL DEL SYNCHRONOUS supprimer un bloc déterminé de la pending queue.

BCFB KL NEXT SYNC Au suivant.

BCFE KL DO SYNC exécuter routine exent.

BD01 KL DONE SYNC routine event terminé.

BD04 KL EVENT DISABLE Verrouillage des évènements normalement simultanés. Les évènements urgents simultanés ne sont pas verrouillés.

BD07 KL EVENT ENABLE autoriser évènements simultanés normaux.

BD0A KL DISARM EVENT verrouiller bloc event (compteur negatif).

BD0D KL TIME PLEASE Combien de temps s'est-il écoulé?
BD10 KL TIME SET Fixer le temps sur valeur indiquée.

BD13 MC BOOT PROGRAM restaure le système d'exploitation et transmet la commande à une routine en (hl).
BD16 MC START PROGRAM initialisation du système et appel d'un programme.
BD19 MC WAIT FLYBACK attendre retour du faisceau.
BD1C MC SET MODE fixer mode écran.
BD1F MC SCREEN OFFSET fixer offset écran.
BD22 MC CLEAR INKS fixer bord écran et inks sur une couleur.
BD25 MC SET INKS fixer couleur pour toutes les inks.
BD28 MC RESET PRINTER réinitialisation du point de branchement indirect pour l'imprimante.
BD2B MC PRINT CHAR imprimer caractère si possible.
BD2E MC BUSY PRINTER imprimante encore occupée?
BD31 MC SEND PRINTER imprimer caractère (attendre jusqu'à ce que ce soit possible).
BD34 MC SOUND REGISTER fournir des données au Sound Controller.

BD37 JUMP RESTORE initialiser tous les vecteurs de saut.

BD3A KM SET STATE
BD3D KM VIDER BUFFER
BD40 TXT FLAG CURSEUR ACTUEL VERS ACCU
BD43 GRA NN
BD46 GRA SAUVER PARAMETRES
BD49 GRA SAUVER PARAMETRES MASQUE
BD4C GRA SAUVER PARAMETRES MASQUE
BD4F GRA CONVERTIR COORD. coordonnées logiques en coordonnées physiques.
BD52 GRA FILL
BD55 SCR MODIFIER DEBUT ECRAN
BD58 MC AFFECTATION DE CARACTERES
BD5B KL FIXER CONFIGURATION RAM

Vecteurs BASIC.

BD5E	EDIT
BD61	FLO COPIER VARIABLE DE (DE) VERS (HL)
BD64	FLO ENTIER VERS VIRGULE FLOTTANTE
BD67	FLO VALEUR 4 OCTETS VERS FLO
BD6A	FLO FLO VERS ENTIER
BD6D	FLO FLO VERS ENTIER
BD70	FLO FIX
BD73	FLO INT
BD76	FLO
BD79	FLO MULTIPLIER NOMBRE PAR 10^A
BD7C	FLO ADDITION
BD7F	FLO RND
BD82	FLO SOUSTRACTION
BD85	FLO MULTIPLICATION
BD88	FLO DIVISION
BD8B	FLO ALLER CHERCHER DERNIERE VALEUR RND
BD8E	FLO COMPARAISON
BD91	FLO CHANGEMENT DE SIGNE
BD94	FLO SGN
BD97	FLO DEG/RAD
BD9A	FLO PI
BD9D	FLO SQR
BDA0	FLO ELEVATION A LA PUISSANCE
BDA3	FLO LOG
BDA6	FLO LOG10
BDA9	FLO EXP
BDAC	FLO SIN
BDAF	FLO COS
BDB2	FLO TAN
BDB5	FLO ATN
BDB8	FLO VALEUR 4 OCTETS VERS FLO
BDBB	FLO RND INIT
BDBE	FLO SET RND SEED

INDIRECTIONS.

BDCD	TXT DRAW/UNDRAW CURSOR fixation/suppression du curseur.
BDD0	TXT DRAW/UNDRAW CURSOR fixation/suppression du curseur
BDD3	TXT WRITE CHAR écrire un caractère sur l'écran.
BDD6	TXT UNWRITE CHAR lire un caractère de l'écran.
BDD9	TXT OUT ACTION sortie d'un caractère sur l'écran ou exécution d'un code de commande?
BDDC	GRA PLOT représenter un point sur l'écran.
BDDF	GRA TEST fournit l'ink de la position graphique actuelle.
BDE2	GRA LINE dessin d'une ligne.
BDE5	SCR READ lecture d'un pixel et décodage d'une ink.
BDE8	SCR WRITE écrire pixel(s).
BDEB	SCR CLEAR vidage de l'écran.
BDEE	KM TEST BREAK ESC, SHIFT et CTRL entraînent une réinitialisation totale du système.
BDF1	MC WAIT PRINTER envoyer un caractère à l'imprimante; si celle-ci n'est pas prête attendre une période de délai.
BDF4	KM UPDATE KEY STATE MAP

2.2 La RAM du système d'exploitation

Voici une liste de la RAM du système d'exploitation, pour autant que nous ayons réussi à découvrir la signification des différentes adresses.

Avant que vous n'utilisiez cette liste, il serait bon que vous réfléchissiez bien aux conséquences de la manipulation de ces adresses. Dans le cas contraire il se pourrait que vous ne détruisiez pas que des flags ou des pointeurs sans signification mais aussi des choses beaucoup plus importantes comme par exemple la table de saut de TEXT SCREEN.

2.2.1 La RAM du système d'exploitation du CPC 664

B82D	KL Start Int Pending Queue
B831	KL div. flags pour routine int.
B832	KL sp save
B8B4	KL Timer low
B8B6	KL Timer high
B8B8	KL Timerflag
B8B9	KL Start Frame Fly Chain
B8BB	KL Start Fast Ticker Chain
B8BD	KL Start Ticker Chain
B8BF	KL Count for Ticker
B8C0	KL Start Sync Pending Queue
B8C2	KL priorité event actuel
B8C3	KL instruction à exécuter
B8D5	KL ROM d'extension actuelle
B8D6	KL entrée ROM actuelle
B8D8	KL configuration ROM actuelle

B7C3	SCR curr. Screen Mode
B7C4	SCR position sur une ligne
B7C6	SCR High Byte Screen Start
B7C7	SCR Write Indirection
B7D2	SCR Flash Periods
B7D3	SCR Flash Period 1. couleur
B7D4	SCR mémoire couleurs 2èmes couleurs
B7E5	SCR mémoire couleurs 1ères couleurs
B7F6	SCR flag jeu de couleurs actuel
B7F8	SCR curr. Flash Period
B7F9	SCR Event Block: Set Inks

B6B5	TXT fenêtre écran actuelle
B6B6	TXT début paramètres fenêtre 0
B726	TXT position curseur actuelle (Row, Col)
B728	TXT flag fenêtre (0=écran fixé)
B729	TXT fenêtre actuelle haut
B72A	TXT fenêtre actuelle gauche
B72B	TXT fenêtre actuelle bas
B72C	TXT fenêtre actuelle droite
B72D	TXT act. Roll Count
B72E	TXT flag curseur actuel
B72F	TXT pen actuel
B730	TXT paper actuel
B731	TXT act. mode fond
B733	TXT Graph Chare Write Mode (0=disable)
B734	TXT 1er caractère User Matrix
B736	TXT Adr. User Matrix
B758	TXT compteur de caractères Control Buffer
B759	TXT Start Control Buffer
B763	TXT table de saut caractère de commande

B693	GRA origine X
B695	GRA origine Y
B697	GRA act. coord. X

B699 GRA act. coord. Y
B69B GRA coord. X GRA fenêtre gauche
B69D GRA coord. X GRA fenêtre droite
B69F GRA coord. Y GRA fenêtre haut
B6A1 GRA coord. Y GRA fenêtre bas
B6A3 GRA Pen
B6A4 GRA Paper
B6A5 GRA buffer de calcul coordonnées X
B6A7 GRA buffer de calcul coordonnées Y

B628 KM Exp. String Pointer
B62A KM Put Back Buffer
B62B KM Adr. Start Exp. Buffer
B62D KM Adr. fin Exp. Buffer
B62F KM Adr. début buffer d'extension libre
B631 KM Shift Lock State
B632 KM Caps Lock State
B633 KM Delay
B635 KM Key State Map
B637 KM Key 16...23
B62B KM Joystick 1
B63E KM Joystick 0
B63F KM touches enfoncées pendant examen
B649 KM Multihit contr. à B63F
B657 KM Break Event Block
B68B KM Adr. Key Translation Table
B68D KM Adr. Key SHIFT Table
B68F KM Adr. Key CTRL Table
B691 KM Adr. de la table de répétition

B1ED SOUND ancienne activité Sound (après HOLD)
B1EE SOUND act. activité Sound
B1F8 paramètres SOUND canal A
B237 paramètres SOUND canal B
B276 paramètres SOUND canal C
B2A6 SOUND courbes d'enveloppe de volume
B396 SOUND courbes d'enveloppe de note

B118	CAS Cass. Message Flag
B11A	CAS Input Buffer Status
B11B	CAS Adr. Start Input Buffer
B11D	CAS Pointer Input Buffer
B11F	CAS File Header Input
B15F	CAS Output Buffer Status
B160	CAS Adr. Start Output Buffer
B162	CAS Pointer Output Buffer
B164	CAS File Header Output
B1E9	CAS Cass. Speed

B115	EDIT Insert Flag

2.2.2 La RAM du système d'exploitation du CPC 6128

B82D	KL Start Int Pending Queue
B831	KL div. flags pour routine int.
B832	KL sp save
B8B4	KL Timer low
B8B5	KL act. configuration RAM
B8B6	KL Timer high
B8B8	KL Timerflag
B8B9	KL Start Frame Fly Chain
B8BB	KL Start Fast Ticker Chain
B8BD	KL Start Ticker Chain
B8BF	KL Count for Ticker
B8C0	KL Start Sync Pending Queue
B8C2	KL priorité event actuel
B8C3	KL instruction à exécuter
B8D6	KL ROM d'extension actuelle
B8D7	KL entrée ROM actuelle
B8D9	KL configuration ROM actuelle

B7C3	SCR curr. Screen Mode
B7C4	SCR position sur une ligne
B7C6	SCR High Byte Screen Start
B7C7	SCR Write Indirection
B7D2	SCR Flash Periods
B7D3	SCR Flash Period 1. couleur
B7D4	SCR mémoire couleurs 2èmes couleurs
B7E5	SCR mémoire couleurs 1ères couleurs
B7F6	SCR flag jeu de couleurs actuel
B7F8	SCR curr. Flash Period
B7F9	SCR Event Block: Set Inks

B6B5	TXT fenêtre écran actuelle
B6B6	TXT début paramètres fenêtre 0
B726	TXT position curseur actuelle (Row, Col)
B728	TXT flag fenêtre (0=écran fixé)
B729	TXT fenêtre actuelle haut
B72A	TXT fenêtre actuelle gauche
B72B	TXT fenêtre actuelle bas
B72C	TXT fenêtre actuelle droite
B72D	TXT act. Roll Count
B72E	TXT flag curseur actuel
B72F	TXT pen actuel
B730	TXT paper actuel
B731	TXT act. mode fond
B733	TXT Graph Chare Write Mode (0=disable)
B734	TXT 1er caractère User Matrix
B736	TXT Adr. User Matrix
B758	TXT compteur de caractères Control Buffer
B759	TXT Start Control Buffer
B763	TXT table de saut caractère de commande

B693	GRA origine X
B695	GRA origine Y
B697	GRA act. coord. X

B699	GRA act. coord. Y
B69B	GRA coord. X GRA fenêtre gauche
B69D	GRA coord. X GRA fenêtre droite
B69F	GRA coord. Y GRA fenêtre haut
B6A1	GRA coord. Y GRA fenêtre bas
B6A3	GRA Pen
B6A4	GRA Paper
B6A5	GRA buffer de calcul coordonnées X
B6A7	GRA buffer de calcul coordonnées Y

B628	KM Exp. String Pointer
B62A	KM Put Back Buffer
B62B	KM Adr. Start Exp. Buffer
B62D	KM Adr. fin Exp. Buffer
B62F	KM Adr. début buffer d'extension libre
B631	KM Shift Lock State
B632	KM Caps Lock State
B633	KM Delay
B635	KM Key State Map
B637	KM Key 16...23
B62B	KM Joystick 1
B63E	KM Joystick 0
B63F	KM touches enfoncées pendant examen
B649	KM Multihit contr. à B63F
B657	KM Break Event Block
B68B	KM Adr. Key Translation Table
B68D	KM Adr. Key SHIFT Table
B68F	KM Adr. Key CTRL Table
B691	KM Adr. de la table de répétition

B1ED	SOUND ancienne activité Sound (après HOLD)
B1EE	SOUND act. activité Sound
B1F8	paramètres SOUND canal A
B237	paramètres SOUND canal B
B276	paramètres SOUND canal C
B2A6	SOUND courbes d'enveloppe de volume
B396	SOUND courbes d'enveloppe de note

B118	CAS Cass. Message Flag
B11A	CAS Input Buffer Status
B11B	CAS Adr. Start Input Buffer
B11D	CAS Pointer Input Buffer
B11F	CAS File Header Input
B15F	CAS Output Buffer Status
B160	CAS Adr. Start Output Buffer
B162	CAS Pointer Output Buffer
B164	CAS File Header Output
B1E9	CAS Cass. Speed

B115	EDIT Insert Flag

2.3 Utilisation des routines du système d'exploitation

Le CPC contient plusieurs centaines de routines ou fonctions dont certaines sont très utiles et parfaitement utilisables par les programmeurs. On trouve par exemple de telles routines pour l'interrogation du clavier, pour sortir un caractère sur l'écran, pour gérer les fenêtres ou pour commander l'imprimante.
Malgré la masse de fonctions dont dispose le système d'exploitation, il y a cependant des choses que le CPC ne sait pas faire de lui-même. C'est ainsi qu'il manque par exemple la possibilité de sortir le contenu de l'écran, texte ou graphisme sur une imprimante connectée au systéme.

Cette possibilité appelée 'Hardcopy', nous allons vous la montrer dans deux exemples. Dans le premier exemple il s'agira d'un hardcopy de texte uniquement, qui fonctionne avec n'importe quelle imprimante connectée. La seconde routine de hardcopy permet l'impression de tous les caractères, y compris les caractères graphiques du CPC. Les images réalisées en graphisme haute résolution peuvent également être imprimées avec cette routine. Nous avons choisi comme imprimante la NLQ 401. Cette imprimante bon marché est, en ce qui concerne son jeu de caractères de commande, étonnamment compatible avec les imprimantes Epson MX/RX/FX. Les deux programmes tournent donc également sans adaptation sur des imprimantes Epson (et sur toutes les autres imprimantes compatibles).

A la fin de ce chapitre, vous ne trouverez pas uniquement deux routines de hardcopy rapides mais vous aurez également une première approche des routines du système d'exploitation.

Pour sortir le contenu de l'écran sur une imprimante connectée, il faut faire lire les caractères ligne par ligne sur l'écran et les sortir. Du fait de la structure spéciale de la Ram vidéo, il n'est malheureusement pas possible de lire les caractères directement.

A travers le 'détour' par une routine du système d'exploitation, il est cependant possible de déterminer quel caractère se trouve dans l'emplacement actuel du curseur.

Cette routine (TXT RD CHAR, &BB60) transmet le caractère dans l'accumulateur et met le flag carry lorsqu'un caractère a été trouvé. Si par contre aucun caractère du jeu de caractères du CPC ne figure dans l'emplacement du curseur, alors l'accumulateur contient 0 et le flag carry est nul.

Il faut en outre une routine qui nous permette de positionner le curseur, de façon à ce que nous puissions lire les caractères les uns après les autres. Cette fonction est exécutée par TXT SET CURSOR, &BB75. Lorsque cette adresse est appelée, le contenu du registre H est interprété comme colonne et celui de L comme ligne. L'emplacement d'écriture suprérieur gauche peut donc être ainsi adressé par &0101.

Il se pose ici cependant une petit problème. Après que nous ayons fait parcourir toute la surface de l'écran à notre curseur, avec l'interrogation de l'écran, il faudrait qu'il revienne ensuite dans son emplacement initial. Il nous faut donc pour cela, avant le premier positionnement du curseur, déterminer et ranger l'emplacement du curseur.
Cela peut se faire grâce à TXT GET CURSOR, &BB78. Après avoir appelé TXT GET CURSOR le double registre HL contient la position actuelle du curseur. Il nous faut ranger cette valeur et la restaurer à la fin du hardcopy.

Les caractères obtenus grâce à TXT RD CHAR doivent être sortis sur l'imprimante. Nous pouvons utiliser à cet effet MC SEND PRINTER dont l'entrée est en &BD31. Le caractère figurant dans l'accumulateur est sorti sur le port d'imprimante avec tous les signaux handshake nécessaires.

MC SEND PRINTER attend toutefois que l'imprimante soit prête à recevoir. C'est MC BUSY PRINTER, &BD2E, qui nous permet de constater si c'est le cas. Si l'imprimante n'est pas prête à recevoir, si elle n'est pas allumée ou si elle n'est même pas connectée, MC BUSY PRINTER revient avec un flag carry mis. Dans ce cas, elle doit être appelée à nouveau, jusqu'à ce que le flag carry soit annulé. Le caractère voulu peut alors être sorti.

Il peut cependant également arriver qu'un hardcopy une fois lancé ne doive pas être imprimé jusqu'au bout. L'opération peut être interrompue en appuyant sur la touche 'DEL'. Mais pour cela, il nous faut pouvoir examiner si cette touche est enfoncée. Si KM TEST KEY, &BB1E, est appelée avec un code de touche valable dans l'accumulateur, après exécution de cette routine, le flag zéro est nul si la touche correspondante est enfoncée. Sinon le flag zéro est mis.

Ainsi avons-nous en fait toutes les routines système nécessaires pour écrire une routine de hardcopy. Mais nous nous rendrons compte au plus tard lorsque nous aurons commencé à écrire notre programme, que nous ne savons absolument pas si, au moment du hardcopy, il s'agit de représenter 20, 40 ou 80 caractères par ligne.
Bon, on pourrait décider que ce hardcopy ne fonctionne qu'en mode d'écran x. Mais ce serait une limitation peu élégante.

SCR GET MODE avec entrée en &BC11 nous communique avec l'accumulateur et les deux flags carry et zéro, dans quel mode écran le CPC se trouve actuellement. Nous pouvons ainsi réaliser un hardcopy avec le nombre de caractères qui convient, en fonction des informations ainsi obtenues.

Mais venons-en maintenant au programme lui-même. Les lecteurs n'ayant pas d'assembleur peuvent utiliser le programme Basic imprimé à la fin de ce chapitre. Il contient les deux programmes de hardcopy en lignes de Data.

```
     BB78    GETCRS    EQU   #BB78
     BB75    SETCRS    EQU   #BB75
     BB60    RDCHAR    EQU   #BB60
     BD2E    TSTPTR    EQU   #BD2E
     BD31    PRTCHR    EQU   #BD31
     BC11    GETMOD    EQU   #BC11
     BB1E    TSTKEY    EQU   #BB1E

A100 CD78BB            CALL  GETCRS      ;ranger ancienne position curseur
A103 2264A1            LD    (OLDPOS),HL
```

```
A106  CD11BC           CALL  GETMOD         ;chercher mode ecran
A109  17               RLA                  ;nombre de caracteres/20
A10A  3263A1           LD    (MODE),A       ;et ranger
A10D  210101           LD    HL,#0101       ;dans angle supérieur gauche
A110  2266A1           LD    (CRSPOS),HL    ;le curseur
A113  3A63A1   LL1     LD    A,(MODE)
A116  47               LD    B,A            ;1,2 ou 4 fois
A117  0E14     LOOP    LD    C,20           ;20 caractères par ligne
A119  C5       LLOOP   PUSH  BC
A11A  E5               PUSH  HL
A11B  CD75BB           CALL  SETCRS         ;placer le curseur
A11E  E1               POP   HL
A11F  CD60BB           CALL  RDCHAR         ;et déterminer
A122  C1               POP   BC             ;le caractére
A123  3802             JR    C,GOOD         ;caractère valable?
A125  3E20             LD    A,32           ;sinon sortir
A127  CD58A1           GOOD  CALL  PRTOUT   ;espace
A12A  E5               PUSH  HL
A12B  C5               PUSH  BC
A12C  3E42             LD    A,66           ;ESC enfoncée?
A12E  CD1EBB           CALL  TSTKEY
A131  C1               POP   BC
A132  E1               POP   HL
A133  201C             JR    NZ,EXIT        ;si oui, fin
A135  24       WEITER  INC   H
A136  0D               DEC   C
A137  20E0             JR    NZ,LLOOP       ;20 caractères imprimés?
A139  10DC             DJNZ  LOOP           ;ligne entiére?
A13B  3E0D             LD    A,#0D          ;sortir CR/LF
A13D  CD58A1           CALL  PRTOUT
A140  3E0A             LD    A,#0A
A142  CD58A1           CALL  PRTOUT
A145  2A66A1           LD    HL,(CRSPOS)    ;déterminer
A148  2C               INC   L              ;position curseur
A149  2266A1           LD    (CRSPOS),HL    ;pour ligne suivante
A14C  7D               LD    A,L
A14D  FE1A             CP    26             ;25 lignes imprimées?
A14F  20C2             JR    NZ,LL1
A151  2A64A1   EXIT    LD    HL,(OLDPOS)    ;si oui, restaurer
```

```
A154  CD75BB           CALL  SETCRS  ;ancienne position curseur
A157  C9               RET           ;et retour
A158  C5      PRTOUT   PUSH  BC
A159  CD2EBD  P1       CALL  TSTPTR  ;printer busy?
A15C  38FB             JR    C,P1
A15E  CD31BD           CALL  PRTCHR  ;sortir un caractére
A161  C1               POP   BC
A162  C9               RET
A163  00      MODE     DEFB  0
A164  0000    OLDPOS   DEFW  0000
A166  0000    CRSPOS   DEFW  0000
```

Les commentaires dans le listing devraient rendre le programme facilement compréhensible. La seule particularité est constituée par la méthode de calcul du nombre de caractères à sortir par ligne. C'est pourquoi nous voudrions évoquer cette question brièvement.

Après que nous ayons appelé SCR GET MODE, l'accumulateur contient, suivant le mode, 0, 1 ou 2. En outre les flags carry et zéro ont les états suivants:

Mode 0 = Carry 1, Zero 0
Mode 1 = Carry 0, Zero 1
Mode 2 = Carry 0, Zero 0

L'instruction SLA décale le contenu de l'accumulateur d'un bit vers la gauche. Cela correspond à une multiplication par deux. L'état du flag carry est en outre tranféré dans le bit 0 de l'accumulateur et le bit 7 qui a été 'expulsé' est placé dans le carry.

En mode 0, le 0 qui se trouve dans l'accumulateur subit une rotation. Cela n'a pas d'influence sur le contenu de l'accumulateur. Mais comme le flag carry qui a été mis par SCR GET MODE est transféré dans le bit 0 de l'accumulateur, l'accumulateur contient 1 après cette instruction. Ce 1 a pour effet que une fois 20 caractères seront imprimés par ligne.

En mode 1, l'accumulateur contient un 1, le carry est nul dans ce mode. Après SLA, l'accumulateur contient un 2. Ce sont donc deux fois 20 caractères qui seront sortis par ligne. Le fonctionnement est analogue en mode 2. Le résultat de SLA est un 4 dans l'accumulateur, ce qui entraîne 4 fois 20 caractères par ligne d'impression.

Le principe est quelque peu différent quand il s'agit de produire un hardcopy graphique. Nous ne pouvons pas alors utiliser les routines TXT SET CURSOR et TXT RD CHAR.
Tout d'abord, GRA INITIALISE active le mode graphique. Ensuite, avec GRA GET PAPER nous déterminons le numéro de couleur du fond. Tous les points de l'écran seront comparés à cette valeur. Si la couleur d'un pixel est différente de celle du fond, un point sera produit sur le papier.

Malheureusement, le CPC ne dispose que d'une connexion 7 bits avec l'imprimante. Il en résulte certaines complications.
Cela signifie d'abord que nous pouvons sortir en une fois sur l'imprimante 7 points placés les uns sous les autres. Le graphisme du CPC a en tout une résolution graphique verticale de 200 points. Mais divisé par 7, cela ne donne pas une valeur entière. Il y a donc un reste, c'est-à-dire des lignes de pixels qui devront être traitées d'une façon particulière. Le problème est cependant identique, quel que soit le mode de texte.

La sortie 7 bits pose un autre problème pour la transmission des instructions à l'imprimante. L'activation du graphisme avec ESC L nécessite pour les 640 pixels par ligne une indication qui ne peut être transmise par le CPC. Pour obtenir le nombre voulu de points graphiques sur l'imprimante, la séquence de commande pour l'imprimante est:

```
PRINT #8,CHR$(27);"L";CHR$(128)CHR$(2)
```

Le problème vient de la valeur 128. Exprimé en terme binaire, 128 est un nombre dont le huitième bit (le bit 7) est mis. Tous les autres bits sont nuls. Si nous envoyions cette valeur sur l'imprimante, celle-ci ne recevrait qu'un 0, puisque le huitième bit est utilisé comme strobe et n'est pas sorti vers l'imprimante.

Nous avons contourné ce problème de façon pas très élégante, en ne sortant horizontalement que 639 points. C'est certes un point de moins qu'il n'y en a sur l'écran, mais nous réduisons ainsi la première valeur à transmettre à 127 (maximum).

Avant que nous n'en venions maintenant au listing du hardcopy graphique, il nous faut encore relever une particularité.
Bien que l'écran ne représente physiquement que 200 lignes de grille, toutes les routines graphiques du CPC raisonnent à partir d'une résolution graphique de 400 points. Il en résulte un meilleur rapport entre les directions X et Y que si l'on ne comptait que les deux lignes véritablement existantes.
La conséquence est facile à observer si vous essayez par exemple le programme de dessin d'un cercle qui vous est proposé dans le manuel du CPC. Vous voyez en effet que le cercle est presque rond. Sans cette correction, c'est une ellipse allongée dans le sens de la largeur qui serait produite.
Cette correction doit également figurer dans notre hardcopy, mais sous une forme exactement contraire. Nous devons également déterminer les coordonnées graphique dans la grille de 400x640 points, mais sur l'imprimante, nous ne sortons que 200 points vertcicalement, pour ne pas avoir de gaspillages trop importants.

```
.COPYRIGHT 1985 MICRO-APPLICATION.
.DAMS *.*.

                 ;
                 ; Hardcopy graphique pour DMP 2000
                 ;     et compatibles epson
                 ;
                          ORG   #a000
                 ;
                 grinit   EQU   #bbba
                 getpap   EQU   #bbe7
                 tstpoi   EQU   #bbf0
                 printo   EQU   #bd2b
                 tstptr   EQU   #bd2e
                 tstkey   EQU   #bb1e
                 ;
                          ENT   $
#A000 CDBABB              CALL grinit            ;Activer le mode graphique.
#A003 CDE7BB              CALL getpap            ;Determiner couleur du fond.
#A006 32B8A0              LD   (paper),a
#A009 CD6CA0              CALL initp             ;Fixer imprimante sur 7/72.
#A00C 218F01              LD   hl,399            ;Nous commencons.
#A00F 22B9A0              LD   (ymerk),hl        ;l'impression
#A012 110000              LD   de,0              ;en haut et a gauche
#A015 3E07                LD   a,7               ;mais avec malheureusement
#A017 32BBA0              LD   (aiguille),a      ; 7 aiguilles.
#A01A CD77A0     lloop    CALL prtesc            ;Sequence ESC pour graphisme.
#A01D 0E00       lll      LD   c,0               ;C contient modele de bits
#A01F 3ABBA0              LD   a,(aiguille)      ;pour l'imprimante.
#A022 47                  LD   b,a               ;B = compteur de lignes.
#A023 E5         bytlp    PUSH hl
#A024 D5                  PUSH de
#A025 C5                  PUSH bc
#A026 CDF0BB              CALL tstpoi            ;Determiner la couleur du
#A029 C1                  POP  bc                ;pixel coord (hl,de).
#A02A D1                  POP  de
#A02B 21B8A0              LD   hl,paper
#A02E BE                  CP   (hl)              ;Couleur pixel = couleur fond ?
#A02F E1                  POP  hl
#A030 37                  SCF                    ;Si pixel <> papier, alors
#A031 2001                JR   nz,dot            ;mettre carry flag, sinon
```

```
#A033 A7                AND  a             ;annuler carry flag.
#A034 CB11      dot     RL   c             ;Inserer carry dans C.
#A036 2B                DEC  hl
#A037 2B                DEC  hl            ;Hl=HL-2 => point suivant,
#A038 10E9              DJNZ bytlp         ;et le tout 7 fois.
#A03A CDAAA0            CALL test          ;Transferer dans accu.
#A03D 79                LD   a,c           ;Traitement special du dernier.
#A03E CDA1A0            CALL print
#A041 13                INC  de            ;Caractere suivant.
#A042 E5                PUSH hl
#A043 217F02            LD   hl,639
#A046 37                SCF
#A047 ED52              SBC  hl,de
#A049 E1                POP  hl
#A04A 3805              JR   c,nxtrow
#A04C 2AB9A0            LD   hl,(ymerk)
#A04F 18CC              JR   lll
#A051 23        nxtrow  INC  hl
#A052 7C                LD   a,h
#A053 B5                OR   l             ;Derniere ligne ?
#A054 C8                RET  z
#A055 2B                DEC  hl
#A056 110000            LD   de,0          ;Preparation de la prochaine
#A059 22B9A0            LD   (ymerk),hl    ;ligne d'impression.
#A05C 3E07              LD   a,7
#A05E BD                CP   l             ;Derniere ligne de 7.
#A05F 20B9              JR   nz,lloop
#A061 7C                LD   a,h
#A062 B4                OR   h
#A063 20B5              JR   nz,lloop
#A065 3E04              LD   a,4           ;alors plus que 4 lignes.
#A067 32BBA0            LD   (aiguille),a
#A06A 18AE              JR   lloop
#A06C 3E1B      initp   LD   a,27          ;Pour DMP 2000 et epson,
#A06E CDA1A0            CALL print
#A071 3E31              LD   a,49          ; ESC "1".
#A073 CDA1A0            CALL print
#A076 C9                RET
#A077 E5        prtesc  PUSH hl
#A078 3E42              LD   a,66
```

```
#A07A CD1EBB              CALL tstkey          ;Touche ESC pressee ?
#A07D E1                  POP  hl
#A07E 2802                JR   z,nokey         ;Si non, passer.
#A080 E1                  POP  hl              ;Depiler pour
#A081 C9                  RET                  ;atteindre le RET.
#A082 3E0D      nokey     LD   a,13            ;Retour chariot
#A084 CDA1A0              CALL print
#A087 3E0A                LD   a,10            ;Saut de ligne.
#A089 CDA1A0              CALL print
#A08C 3E1B                LD   a,27            ;Sequence ESC "L" 127 2
#A08E CDA1A0              CALL print           ;pour graphisme
#A091 3E4C                LD   a,76            ;avec 639 points.
#A093 CDA1A0              CALL print
#A096 3E7F                LD   a,127
#A098 CDA1A0              CALL print
#A09B 3E02                LD   a,2
#A09D CDA1A0              CALL print
#A0A0 C9                  RET
#A0A1 CD2EBD    print     CALL tstptr          ;Imprimante busy ?
#A0A4 38FB                JR   c,print
#A0A6 CD2BBD              CALL printo          ;Imprimer un caractere.
#A0A9 C9                  RET
#A0AA 3ABBA0    test      LD   a,(aiguille)
#A0AD FE07                CP   7
#A0AF C8                  RET  z
#A0B0 AF                  XOR  a               ;Traitement special
#A0B1 CB11                RL   c               ;des 4 dernieres lignes
#A0B3 CB11                RL   c               ;de pixel.
#A0B5 CB11                RL   c
#A0B7 C9                  RET
                ;
#A0B8 00        paper     DEFB 0
#A0B9 0000      ymerk     DEFW 0
#A0BB 00        aiguille  DEFB 0

Pass 2: 0 Errors

End of code:#A0BC
Executes:#A000
```

```
100 ' Hardcopy de texte pour cpc
120 '   Le hardcopy doit etre appele par
130 '            CALL &A100
299 S=0
300 FOR i=&A100 TO &A162
310 READ byte:POKE i,byte:s=s+byte:NEXT
320 DATA &CD,&78,&BB,&22,&64,&A1,&CD,&11
325 DATA &BC,&17,&32,&63,&A1,&21,&01,&01
330 DATA &22,&66,&A1,&3A,&63,&A1,&47,&0E
335 DATA &14,&C5,&E5,&CD,&75,&BB,&E1,&CD
340 DATA &60,&BB,&C1,&38,&02,&3E,&20,&CD
345 DATA &58,&A1,&E5,&C5,&3E,&42,&CD,&1E
350 DATA &BB,&C1,&E1,&20,&1C,&24,&0D,&20
355 DATA &E0,&10,&DC,&3E,&0D,&CD,&58,&A1
360 DATA &3E,&0A,&CD,&58,&A1,&2A,&66,&A1
365 DATA &2C,&22,&66,&A1,&7D,&FE,&1A,&20
370 DATA &C2,&2A,&64,&A1,&CD,&75,&BB,&C9
375 DATA &C5,&CD,&2E,&BD,&38,&FB,&CD,&31
380 DATA &BD,&C1,&C9
385 '
390 IF s<>11873 THEN PRINT "erreur dans HC texte":END
400 PRINT "chargement de HC texte correct":END
```

```
100 ' Hardcopy graphique pour DMP 2000
110 '     et compatibles epson
120 ' Le hardcopy doit etre appele par 'call &A000'
130 '
140 DATA CD,BA,BB,CD,E7,BB,32,B8,A0,CD,6C,A0,21,8F,01,22
150 DATA B9,A0,11,00,00,3E,07,32,BB,A0,CD,77,A0,0E,00,3A
160 DATA BB,A0,47,E5,D5,C5,CD,F0,BB,C1,D1,21,B8,A0,BE,E1
170 DATA 37,20,01,A7,CB,11,2B,2B,10,E9,CD,AA,A0,79,CD,A1
180 DATA A0,13,E5,21,7F,02,37,ED,52,E1,38,05,2A,B9,A0,18
190 DATA CC,23,7C,B5,C8,2B,11,00,00,22,B9,A0,3E,07,BD,20
200 DATA B9,7C,B4,20,B5,3E,04,32,BB,A0,18,AE,3E,1B,CD,A1
210 DATA A0,3E,31,CD,A1,A0,C9,E5,3E,42,CD,1E,BB,E1,28,02
220 DATA E1,C9,3E,0D,CD,A1,A0,3E,0A,CD,A1,A0,3E,1B,CD,A1
230 DATA A0,3E,4C,CD,A1,A0,3E,7F,CD,A1,A0,3E,02,CD,A1,A0
240 DATA C9,CD,2E,BD,38,FB,CD,2B,BD,C9,3A,BB,A0,FE,07,C8
250 DATA AF,CB,11,CB,11,CB,11,C9,00,00,00,00
260 '
270 MEMORY  40959
280 s=0
290 FOR i= 40960 TO 41147
300 READ b$:b=VAL("&"+b$):POKE i,b:s=s+b
310 NEXT
320 IF s<> 23051 THEN PRINT "erreur en DATA"
330 END
```

2.4 Le traitement des interruptions dans le système d'exploitation

La possibilité la plus rapide et la plus puissante de réagir à l'intérieur du système d'exploitation à certains évènements est sans doute la technique des interruptions.
Vous savez certainement ce que c'est. Sinon, voici l'essentiel de ce qu'on peut dire à ce sujet:

Une interruption est en général un évènement d'ordre électronique qui informe un programme en train de tourner qu'il vient de se produire. En fonction de cet évènement, le logiciel doit entreprendre des actions correspondantes et ce, le plus vite possible, suivant le niveau d'urgence. Une telle action sera par exemple le scrolling de l'écran pendant la phase sombre du rayon électronique, de façon à ce que l'image soit le plus nette possible.

Cette technique d'interruption présente l'avantage de n'interrompre le déroulement du programme que lorsqu'il y a vraiment une action à effectuer, de sorte que le logiciel n'est pas constamment obligé de contrôler s'il se passe quelque chose ou non.
Il y a naturellement de nombreuses possibilités pour intégrer une telle fonction dans un système d'exploitation mais nous devons reconnaître que nous n'avions jamais encore rencontré une variante du type de celle qui fonctionne sur le CPC.

Il s'agit ici d'un mélange raffiné d'interruption hardware (interruption lorsque nécessaire) et de polling (examen régulier de ce qui se passe). Le programmeur de la routine correspondante décide du niveau d'urgence d'une 'demande'. En clair:
Il n'y a qu'une seule interruption dans la machine, le timer (appelé fast ticker dans le système), qui produit une interruption tous les 300èmes de seconde. Tout le reste en découle, comme vous allez voir.

Il est maintenant temps d'introduire quelques concepts que vous rencontrerez souvent à partir de maintenant, y compris dans le commentaire de la ROM.

1. EVENT signifie tout simplement évènement. Comprenez qu'il s'agit d'une sorte d'interruption commandée par logiciel.

2. FRAME FLYBACK n'est rien d'autre que le retour déjà évoqué du rayon de l'écran, ce qui se produit tous les 50èmes de seconde.

3. TICKER est un multiple du fast ticker qui apparaît également tous les 50èmes de seconde.

Le tout est traité de façon à ce que le programmeur, donc éventuellement vous même, puisse définir quelles routines de son programme devront être appelées automatiquement, sans aucune intervention supplémentaire, et avec quelle fréquence elles devront être appelées au moment du flyback, ticker ou même fast ticker. Comme préparation, il suffit, outre quelques petits détails, de communiquer une fois l'adresse de cette ou de ces routines.
Cette information à préparer s'appelle EVENT BLOCK. Ici est indiqué avec quelle fréquence et quant la routine doit être appelée, si elle est ou non prioritaire par rapport à d'autres routines, etc...

A l'entrée du Ticker, Fast Ticker ou Frame Fly, le système d'exploitation regarde s'il y a des Event blocks correspondants. Si oui, ils sont appelés, en fonction de leur degré de priorité. Certains event blocks existent en permanence, comme par exemple l'action qui consiste à alimenter le registre de couleur au moment du Frame Fly.

Les blocs affectés à un évènement déterminé sont également reliés ensemble par le pointeur, de sorte que le système d'exploitation peut osciller de l'un à l'autre. Il est donc sans importance de savoir à quelle adresse figure un tel bloc, tant qu'il se trouve dans les 32K centraux de la RAM. Cette petite réserve doit être faite car cette zone est la seule à laquelle il soit possible d'accéder en permanence, indépendamment de la configuration de la ROM.

Si un tel bloc doit être exécuté, il est rangé dans ce qu'on appelle Pending Queue. Ce procédé est appelé Kicking.

La Pending Queue est traitée à la fin de la routine d'interruption propre du système. Vous vous dites certainement qu'un bloc existant doit naturellement être exécuté. Pourquoi faut-il donc le ranger dans une queue?
En fait, les choses ne sont pas aussi simples car vous avez tout à fait la possibilité de suspendre le traitement d'un bloc pour un certain temps, sans que vous ayez à l'extraire de la queue primaire; ceci est d'ailleurs très facile à réaliser avec les event blocks de la ticker queue.

A propos: ne croyez pas qu'il n'y ait que cette interruption dans l'ordinateur. Les fanas de l'électronique ont tout à fait la possibilité de produire une interruption à travers le bus d'extension (asynchron), mais il faut bien sûr qu'il y ait une routine correspondante qui puisse 'kicker' l'event block correspondant.

Devenons plus concret. Que faut-il faire lorsque vous voulez utiliser ce mécanisme?
Il faut bien sûr commencer par créer un event block dont la structure est définie ci-après. La partie suivante est commune à toutes sortes d'évènements:

Octets 0+1 adresse de chaîne pour la pending queue. Ce champ ne doit être alimenté que par le système d'exploitation!

Octet 2 compteur
Tant que le compteur est > 0, le bloc reste dans la pending queue, c'est-à-dire que la routine est exécutée jusqu'à ce qu'il soit égal à 0.

Si le compteur est < 0 (c'est-à-dire > 127), le bloc reste dans la chaîne correspondante (ticker etc...). Le kicking ne conduit pas non plus dans ce cas à une exécution de la routine, alors que cela aurait normalement pour effet d'augmenter le compteur et donc de provoquer un saut à la prochaine occasion.

Octet 3 classe
Bit 0 = 1 = L'adresse de saut est une Near Adress, c'est-à-dire qu'elle se trouve dans la RAM centrale ou dans la ROM inférieure.
Bit 0 = 0 = L'adresse de saut est une Far Address, donc à rechercher dans la ROM supérieure.
Les bits 1-4 déterminent la priorité.
Bit 5 doit toujours valoir 0 !
Bit 6 = 1 = Express. Les express events ont une priorité supérieure à celle des évènements normaux de la plus grande priorité.
Bit 7 = 1 = Asynchron Event. Ces évènements n'ont pas de file d'attente et ils sont rangés immédiatement dans l'interrupt pending queue lors du kicking (KL EVENT). S'il s'agit même d'un express, cette routine est exécutée immédiatement, sinon seulement à la fin de la routine d'interruption.
Attention: la routine pour les évènements asynchrones doit absolument se trouver dans la RAM centrale!

Octets 4+5 Adresse de la routine

Octet 6 Rom Select, si l'adresse de saut est du type Far, sinon inutilisé.

Octet 7 Ici commence le champ de l'utilisateur qui peut être aussi long que souhaité. Il peut servir à la transmission de paramètres à la routine. Lors de l'appel d'une event routine, hl contient l'adresse de l'octet 5 de l'event block, s'il s'agit d'une near adress, sinon l'adresse de l'octet 6.
Ceci permet de créer plusieurs blocs pour une même routine qui peut déterminer, en fonction des paramètres, par quel bloc elle a été appelée.

Suivant le type d'évènement, Ticker, Fast Ticker ou Frame Fly, deux ou six octets sont encore placés avant la partie commune. Dans le cas de Fast Ticker et Frame Fly, ce ne sont que deux octets pour le chaînage (ne pas les modifier!) dans la Fast Ticker List ou la Frame Fly List.

Les six octets pour le Ticker ont la signification suivante:

Octets 0+1 Chaînage pour Ticker List (ne pas modifier!)

Octets 2+3 Tick Count détermine combien de fois un ticker doit apparaître, avant que le bloc ne soit kické une fois.

Octets 4+5 Reload Count indique quelle valeur doit être chargée dans le Tick Count après son écoulement.

Après donc que vous ayez alimenté votre bloc avec ces valeurs, pour autant que vous les connaissiez, (ce devraient être les 5 derniers octets (event count=0) de la partie commune et, pour le ticker, également les compteurs), vous n'avez plus qu'à charger l'adresse de début de votre bloc dans hl, puis, suivant le cas, à appeler la routine KL ADD TICKER, KL ADD FAST TICKER ou KL ADD FRAME FLY.

Pour extraire le bloc de la liste, utilisez les routines KL DEL TICKER, etc... hl devant cette fois également contenir l'adresse du bloc à retirer.

Essayez et observez comment le système d'exploitation procède, car les procédures qui reviennent sans cesse sont également traitées à travers le mécanisme des évènements.

2.5 La Rom du système d'exploitation

Vous avez certainement déjà remarqué, lors de la présentation des vecteurs du système d'exploitation (chapitre 2.1) et de la RAM du système d'exploitation (chapitre 2.2), que les différences entre le CPC 664 et le CPC 6128 se réduisent au minimum. C'est pourquoi nous avons décidé, pour ne pas gaspiller du papier, de concentrer notre présentation sur le CPC 6128. Cela ne veut pas dire que les possesseurs d'un CPC 664 peuvent maintenant refermer le livre et le ranger dans un placard. Simplement, il faudra qu'ils fassent un peu plus attention et qu'ils se demandent si les commentaires imprimés correspondent immédiatement à leur listing de la ROM, ou bien si les adresses sont décalées de quelques octets. Comme vous êtes suffisamment avancé en langage-machine pour vous lancer dans des expériences avec le système d'exploitation, ce travail de transposition ne devrait pas vous poser de problème. Toutefois, si vous ne vous sentez pas suffisamment sûr de vous, nous vous recommandons d'utiliser exclusivement les vecteurs.

Vous allez maintenant trouver, dans les pages suivantes, le commentaire du système d'exploitation du CPC 6128. Ces commentaires sont peu parlants en eux-mêmes. Par contre, si vous produisez un listing grâce au désassembleur imprimé en annexe de ce livre et que vous rapprochez les adresses du listing des adresses indiquées dans les commentaires, vous obtenez un ensemble extrêmement précieux et intéressant.

Ce type de commentaires pour les systèmes d'exploitation est d'ailleurs certainement appelé à devenir la norme, étant donné la complexité croissante de ces structures.

2.5.1 KERNAL (KL)

Le Kernal, comme son nom l'indique est le noyau du système d'exploitation.
C'est ainsi qu'il est responsable de la commande du déroulement des programmes, c'est-à-dire du traitement des interruptions ainsi que des Events, du traitement des Restarts, de la mise en place d'extensions de la Rom et de la commutation entre les diverses configurations de la mémoire.
Les routines liées au mécanisme des Events peuvent être intéressantes pour l'utilisateur.

0000 ********************************** RST 0 RESET ENTRY

Après la mise sous tension du système, le processeur commence ici le traitement du programme. Un appel de RST 0 a pour effet une réinitialisation totale du système.

0000 U ROM disable, Mode 1, reset diviseur
0005 Reset Cont'd

0008 ************************************** RST 1 LOW JUMP

Sert à appeler une routine dans le système d'exploitation ou dans la RAM lui étant parallèle. L'adresse de la routine à appeler doit figurer directement à la suite de l'instruction RST. Comme pour la zone de &0000 à &3FFF, 14 bits d'adresse suffisent, les bits 14 et 15 sont utilisés pour sélectionner entre ROM et RAM. Le bit 15 mis a pour effet de sélectionner la RAM dans la zone de &C000 à &FFFF alors que le bit 14 annulé a pour effet de sélectionner le système d'exploitation.

0008 (0430) RST 1 LOW JUMP CONT'D
000B (042A) KL LOW PCHL CONT'D
000E manipuler adresse de retour
000F correspond à jp (bc)

0010 ************************************* RST 2 SIDE CALL

Sert à appeler une routine dans une ROM d'extension. RST 2 est utilisé quand un programme, connecté sous forme de ROM d'extension, a besoin de plus de 16 K.

0010	(04C3) RST 2 LOW SIDE CALL CONT'D
0013	(04BD) KL SIDE PCHL CONT'D
0016	manipuler adresse de retour
0017	correspond à jp (de)

0018 ************************************* RST 3 FAR CALL

Permet d'appeler une routine n'importe où, en ROM ou en RAM. Il faut pour cela faire suivre l'instruction RST 3 de l'adresse sur deux octets d'un bloc de paramètres (de 3 octets de long). Les deux premiers octets du bloc de paramètres contiennet l'adresse de la routine à appeler. Le troisième octet indique l'état ROM/RAM.

0018	(046D) RST 3 LOW FAR CALL CONT'D
001B	(045F) KL FAR PCHL CONT'D

0020 ************************************* RST 4 RAM LAM

RST 4 vous permet de lire le contenu de la RAM à partir d'un programme en langage machine, quel que soit l'état ROM sélectionné. L'instruction RST 4 remplace pratiquement LD A,(HL); hl doit pour cela contenir l'adresse de la case mémoire à lire.

0020	(056C) RST 4 RAM LAM CONT'D
0023	(0467) KL FAR ICALL CONT'D

0028 ************************************ RST 5 FIRM JUMP

Permet de sauter à une routine du système d'exploitation. L'adresse d'entrée correspondante doit suivre immédiatement l'instruction RST 5.

Avant que le saut à la routine voulue ne soit exécuté, la ROM du système d'exploitation est sélectionnée puis à nouveau déconnectée après abandon de la routine.

0028 (04DB) RST 5 FIRM JUMP CONT'D

0030 ******************************** RST 6 USER RESTART

Die Bytes &0030 bis &0037 stehen dem Benutzer zur Verfügung. Beim Einschalten des Systems ist ein RST 0 voreingestellt.
Les octets &0030 à &0037 sont disponibles pour l'utilisateur. Lors de la mise sous tension du système, un RST 0 est fixé d'avance.

0030 RST 0 vers High Kernel Restore

0038 *************************** RST 7 INTERRUPT ENTRY

entrée pour interruptions hardware.

0038 (03E7) RST 7 INTERRUPT ENTRY CONT'D
003B EXT INTERRUPT

0040 ************************jusqu'ici on copie dans la RAM

0040 L ROM disable

0044 ***************************** Restore High Kernel Jumps

0044 copier 003F
0047 jusqu'à
0048 0000
0049 dans la RAM
004A
004C RST 0 vers
004E 0030
0051 Jump
0054 Block
0057 (copier)

005C ************************************* KL CHOKE OFF

Restaurer kernal, supprimer files d'attente Event et cetera

005D	(configuration ROM actuelle)
0060	(entrée ROM actuelle)
0064	supprimer
0066	RAM firmware
0069	jusqu'à
006B	B8CD
006C	
0071	une ROM était-elle activée?
0072	oui sauter
007C	si hl=0
007D	charger défaut
0080	(ROM d'extension actuelle)
0083	(configuration ROM actuelle)
0086	(entrée ROM actuelle)
0089	charger paramètres
008C	pour RST 3
0095	FAR CALL
0096	dw B8D7

0099 ************************************* KL TIME PLEASE

Combien de temps s'est-il écoulé?

009A	(Timer high)
009E	(Timer low)

00A3 ************************************* KL TIME SET

Fixer le temps sur valeur indiquée.

00A4	charger 0 dans accu et restaurer flags
00A5	(flag timer)
00A8	(Timer high)
00AC	(Timer low)

00B1 ** Scan Events

00B1	Timer low
00B4	update
00B5	Timer
00BA	Port B
00BC	VSYNC?
00BD	non sauter
00BF	(Start Frame Fly Chain)
00C2	octet fort vers accu
00C3	accu = 0?
00C4	accu pas 0 sauter à Kick Event
00C7	(Start Fast Ticker Chain)
00CA	octet fort vers accu
00CB	accu = 0?
00CC	accu pas 0 sauter à Kick Event
00CF	Scan Sound Queues
00D2	Count for Ticker
00D9	Update Key State Map
00DC	(Start Ticker Chain)
00DF	octet fort vers accu
00E0	accu = 0?
00E1	accu 0 sauter
00E2	divers flags pour routine int.
00E5	Ticker Chain doit encore
00E7	être traité
00F2	(Start Int Pending Queue)
00F8	divers flags pour routine int.
010A	(sp save)
010E	Timer low
0114	divers flags pour routine int.
011D	(Start Int Pending Queue)
0120	octet fort vers accu
0121	accu = 0?
0122	accu 0 sauter
0127	(Start Int Pending Queue)
0132	divers flags pour routine int
0135	Ticker Queue pending ?
0137	non sauter

013D traiter Ticker Chain
0142 divers flags pour routine int
0145 octet fort vers accu
0146 encore quelque chose à traiter?
0147 oui sauter
0149 annuler flags
014E restaurer sp

0153 *** Kick Event

0158 KL EVENT
015D KL EVENT
0161 Kick Event

0163 ********************************** KL NEW FRAME FLY

créer et ajouter bloc event

0166 KL INIT EVENT

016A ********************************* KL ADD FRAME FLY

ajouter bloc event.

016A Start Frame Fly Chain
016D Add Event

0170 ********************************** KL DEL FRAME FLY

supprimer bloc event.

0170 Start Frame Fly Chain
0173 Delete Event

0176 ******************************** KL NEW FAST TICKER

créer et ajouter bloc event (voir KL NEW FRAME FLY).

0179 KL INIT EVENT

017D ****************************** KL ADD FAST TICKER

ajouter bloc event (voir KL ADD FRAME FLY).

017D   Start Fast Ticker Chain
0180   Add Event

0183 ******************************* KL DEL FAST TICKER

supprimer bloc event (voir KL DEL FRAME FLY).

0183   Start Fast Ticker Chain
0186   Delete Event

0189 ******************************** traiter Ticker Chain

0189   (Start Ticker Chain)
018C   octet fort vers accu
018D   accu = 0?
018E   accu 0 sauter
01A4   KL EVENT

01B3 ************************************* KL ADD TICKER

ajouter bloc ticker.

01BF   Start Ticker Chain
01C2   Add Event

01C5 ************************************* KL DEL TICKER

supprimer bloc ticker.

01C5   Start Ticker Chain
01C8   Delete Event

01D2 ************************************** KL INIT EVENT

créer bloc event.

01E2 ** KL EVENT

'kick' bloc event.

01E7 Event Cnt >127/<0
01EB Event Cnt >0&<127
01F1 ajouter sync event

0219 ** KL DO SYNC

exécuter routine exent.

021F (0467) KL FAR INCALL CONT'D

0227 ************************************* KL SYNC RESET

annuler sync pending queue

022E ********************************* ajouter sync event

022F priorité vers b
0230 instruction à exécuter
0236 adresse du prochain
0237 bloc event
0238 amener vers de
0240 priorité actuelle > trouvée
0241 priorité?
0242 non sauter

0255 ************************************* KL NEXT SYNC

Au suivant.

0256 (Start Sync Pending Queue)
0259 octet fort vers accu
025A accu = 0?
025B accu 0 sauter
0263 (priorité event actuel)
026B (priorité event actuel)

026E		(Start Sync Pending Queue)

0276 ************************************** KL DONE SYNC

routine event terminée.

0276		(priorité event actuel)
027E		ajouter sync event

0284 ****************************** KL DEL SYNCHRONOUS

supprimer un bloc déterminé de la pending queue.

0284		KL DISARM EVENT
0287		Start Sync Pending Queue
028A		Delete Event

028D *********************************** KL DISARM EVENT

verrouiller bloc event (compteur négatif).

0294 ********************************** KL EVENT DISABLE

Verrouiller les évènements simultanés normaux. Les évènements simultanés urgents ne sont pas verrouillés.

0294		priorité event actuel

029A ****************************á**** KL EVENT ENABLE

autoriser évènements simultanés normaux.

029A		priorité event actuel

02A0 ** KL LOG EXT

ajouter extensions résidentes.

02B1 ********************************* KL FIND COMMAND

chercher instruction dans toutes les zones mémoire ajoutées.

02B1	instruction à exécuter
02B7	(0553) KL ROM OFF & CONFIGURATION SAVE
02DA	(0524) KL PROBE ROM CONT'D
02E4	MC START PROGRAM
02FC	(051F) KL ROM SELECT CONT'D
0307	instruction à exécuter
0323	(052D) KL ROM DESELECT CONT'D

0326 ************************************ KL ROM WALK

trouve et initialise extensions ROM pour que ces ROMs soient disponibles.

0328	KL INIT BACK

0330 ************************************** KL INIT BACK

ajouter extensions ROM.

0330	configuration ROM actuelle
0339	(051F) KL ROM SELECT CONT'D
0351	(ROM d'extension actuelle)
0360	KL LOG EXT
0366	(052D) KL ROM DESELECT CONT'D

0379 *** Add Event

0388 *** Delete Event

0397 ***************KL FIXER CONFIGURATION RAM

Ici est effectuée une commutation entre les différentes configurations RAM du CPC 6128.

0398	sauver registres

0399 configuration RAM actuelle
039E préparation pour Gate-Array
03A0 commutation configuration RAM
03A3 rétablir ancien état des registres

03A6 (0505) KL U ROM ENABLE CONT'D
03A9 (050C) KL U ROM DISABLE CONT'D
03AC (04F7) KL L ROM ENABLE CONT'D
03AF (04FE) KL L ROM DISABLE CONT'D
03B2 (0516) KL ROM RESTORE CONT'D
03B5 (051F) KL ROM SELECT CONT'D
03B8 (0543) KL CURR SELECTION CONT'D
03BB (0524) KL PROBE ROM CONT'D
03BE (052D) KL ROM DESELECT CONT'D
03C1 (0547) KL LDIR CONT'D
03C4 (054D) KL LDDR CONT'D

03C7 ****************************** KL POLL SYNCHRONOUS

Y a-t-il un event de priorité supérieure à celle de l'actuel?

03D6 (Start Sync Pending Queue)
03E0 (priorité event actuel)

03E7 ****************** RST 7 INTERRUPT ENTRY CONT'D

comparez avec RST 7 INTERRUPT ENTRY.

03E9 KL EXT INTERRUPT ENTRY
03F4 L ROM enable
03F6 Scan Events
03FE (divers flags pour routine int.)
0418 fixer ancienne configuration

041E ************************* KL EXT INTERRUPT ENTRY

0423 L ROM disable

042A ******************************** KL LOW PCHL CONT'D

saut en ROM ou RAM basse.

0430 ***************************** RST 1 LOW JUMP CONT'D

comparez avec RST 1 LOW JUMP.

043C	rotation accu quatre fois vers la gauche
0445	(0456) préparer configuration et exécuter saut
0456	placer adresse de saut sur la pile
0458	fixer configuration ROM
045E	exécuter saut préparé

045F ******************************** KL FAR PCHL CONT'D

0467 ******************************* KL FAR ICALL CONT'D

046D ********************** RST 3 LOW FAR CALL CONT'D

comparez avec RST 3 LOW FAR CALL.

047C	ROM# > 252?
047E	oui sauter
0480	ROM d'extension
0482	(connecter)
0484	ROM d'extension actuelle
04A2	L ROM disable
04A4	U ROM enable
04A6	(0456) préparer configuration et exécuter saut
04AF	restaurer
04B0	ancienne
04B1	configuration
04B3	de ROM d'extension
04B5	(ROM actuelle d'extension)

04BD ******************************* KL SIDE PCHL CONT'D

04C3 ********************** RST 2 LOW SIDE CALL CONT'D

comparez avec RST 2 LOW SIDE CALL.

04D5 (configuration ROM actuelle)

04DB *************************** RST 5 FIRM JUMP CONT'D

comparez avec RST 5 FIRM JUMP.

04E3 L ROM enable
04E5 charger adresse de saut
04EB et exécuter saut
04F0 L ROM disable

04F7 ************************* KL L ROM ENABLE CONT'D

connecter ROM inférieure.

04FA L ROM enable
04FC saut à exécution

04FE ************************* KL L ROM DISABLE CONT'D

déconnecter ROM inférieure

0501 L ROM disable
0503 saut à exécution

0505 ************************* KL U ROM ENABLE CONT'D

connecter ROM supérieure.

0508 U ROM enable
050A saut à exécution

050C ************************ KL U ROM DISABLE CONT'D

déconnecter ROM supérieure.

050F U ROM disable
0511 exécution

0516 *************************** KL ROM RESTORE CONT'D

restaurer ancienne configuration ROM.

0517 a contient
0518 l'ancienne
0519 configuration
051D saut à exécution

051F ****************************** KL ROM SELECT CONT'D

sélectionner une ROM supérieure déterminée.

051F (0505) KL U ROM ENABLE CONT'D

0524 ******************************* KL PROBE ROM CONT'D

examiner ROM.

0524 (051F) KL ROM SELECT CONT'D

052D ************************** KL ROM DESELECT CONT'D

restaurer ancienne configuration ROM supérieure.

052F (0516) KL ROM RESTORE CONT'D
0535 ROM d'extension (# in c)
0537 (connecter)
0539 ROM actuelle d'extension

0543 *********************** KL CURR SELECTION CONT'D

Quelle ROM supérieure est activée?

0543 ROM actuelle d'extension

0547 ************************************ KL LDIR CONT'D

LDIR pour ROMs bloquées.

054D (0553) KL ROM OFF & CONFIG.SAVE

054D *********************************** KL LDDR CONT'D

LDDR pour ROMs bloquées.

054D (0553) KL ROM OFF & CONFIG.SAVE

0553 ********************** KL ROM OFF & CONFIG. SAVE

0555 manipuler adresse RET
0556 sauver ancienne configuration sur la pile
0557 ROMs
0559 disable
055D call (hl)
0561 restaurer
0562 ancienne
0563 configuration
0568 manipuler adresse RET

056C **************************** RST 4 RAM LAM CONT'D

Comparez RST 4 RAM LAM.

056F ROMs
0571 disable
0576 aller chercher octet
0578 fixer ancienne configuration

057D ************************************ KL RAM LAM (IX)

correspond à ld a,(ix).

057F ROMs
0581 disable

0583 aller chercher octet
0586 fixer ancienne configuration

2.5.2 MACHINE PACK (MC)

C'est la partie du système d'exploitation qui est la plus proche de la machine.
C'est ici que sont traités les divers interfaces et éléments périphériques tels que PIO et PSG. Cette procédure présente l'avantage qu'en cas de modification éventuelle de l'électronique, seul le MACHINE PACK devra être adapté comme par exemple le BIOS en CP/M.
De ce fait, seules quelques routines peuvent être utilisées souvent.

0591 ** Reset Cont'd

0592 Control
0597 Port A
059C Port C
05A1 Centronics
05A6 Port B
05AA isoler LK4
05AC fin table 60Hz
05AF 50Hz? sauter si pas
05B1 fin table 50Hz
05B7 charger adresse registres vidéo
05BC charger registres vidéo

05C5 *** table 60Hz

3F 28 2E 8E 26 00 19 1E
00 07 00 00 30 00 C0 00

05D5 *** table 50Hz

3F 28 2E 8E 1F 06 19 1B
00 07 00 00 30 00 C0 00

05E5 démarrage à froid
05E8 à adresse de
05EB continuation

05ED ********************************** MC BOOT PROGRAM

restaure le système d'exploitation et transmet la commande à une routine en (hl).

05F1 SOUND RESET
05F5 restaurer
05F8 périphérie
05FA KL CHOKE OFF
0601 KM RESET
0604 TXT RESET
0607 SCR RESET
060A KL U ROM ENABLE CONT'D
060E jp (hl)
0613 MC START PROGRAM
0617 erreur de chargement

061C ********************************* MC START PROGRAM

initialisation complète du système et appel du programme dont adresse de début en hl.

061C rencontre RET après 066F
0620 fixer mode d'interruption 1
0622 sauver contenu des registres
0623 restaurer pointeur palette
0628 reset de la périphérie
062B éventuellement connectée
062D réinitialiser
0630 configuration RAM
0632 Floppy-Motor on/off Flip/Flop
0636 déconnecter moteur disquette
0638 copier &7f9 octets de l'adresse de départ
063B &B100 à l'adresse objet
063E &B101

0641 charger contenu de l'accu dans &B100
0642 exécuter procédure de copie
0644 U ROM off & L ROM on
0647 Sreen Mode 1
0649 restaurer ancien contenu des registres
0652 Restore High Kernel Jumps
0655 JUMP RESTORE
0658 KM INITIALISE
065B SOUND RESET
065E SCR INITIALISE
0661 TXT INITIALISE
0664 GRA INITIALISE
0667 CAS INITIALISE
066A MC RESET PRINTER
066F jp (hl)
0674 initialiser ROM supérieure

0677**démarrage à froid

067A TXT SET CURSOR
067D sortir noms de firme
0680 sortir messages
0683 message initial
0686 sortir messages

0688 ************************************** message initial

0689 128K
068E Microcomputer
069D (v3)
06A4 Copyright
06B0 c1985
06B6 Amstrad
06BE Consumer
06C7 Electronics
06D3 plc
06D9 and
06DD Locomotive

06E8 Software
06F1 Ltd

06F9 message erreur de chargement

06FC ************************************** sortir message

0700 TXT OUTPUT
0703 sortir message

0705 ************************************* message erreur de chargement

0705 ***
0709 PROGRAM
0711 LOAD
0716 FAILED
071E ***

0725 Port B
0728 isoler LK1 ... 3
072A /2
072B noms de firme

0738 *** noms de firme

0738 Arnold
073F Amstrad
0747 Orion
074D Schneider
0757 Awa
075B Solavox
0763 Saisho
076A Triumph
0772 Isp

0776 *** MC SET MODE

fixer mode écran.

0776 Mode>2?
0778 si oui retour
077B restaurer
077D bits mode
077D fixer
0781 nouveau mode

0786 ** MC CLEAR INKS

fixer une couleur pour cadre écran et toutes les inks.

0786 placer contenu de hl sur la pile
0787 puis charger &0000 dans hl
078A six octets plus loin

078C *** MC SET INKS

Sortir couleur de toutes les inks et du bord écran.

078C placer contenu de hl sur la pile
078D puis charger &0001 dans hl
0793 couleur bord
0796 sortir couleur
079A Adresse Ink 0
079C sortir couleur
07A4 charger toutes les mémoires couleur

07AA ** sortir couleur

07AA pointeur de palette
07AD annuler bits 5, 6 et 7 de l'accu
07AF puis mettre bit 6
07B1 couleur

07B4 ************************************* MC WAIT FLYBACK

attendre retour du faisceau.

07B6 Port B
07BA VSYNC?
07BB si non attendre

07C0 ********************************** MC SCREEN OFFSET

fixer offset écran.

07C3 annuler tous les bits sauf 4 et 5
07C8 annuler tous les bits sauf 0 et 1
07CE Video Contr Register 12
07D1 début écran Hi
07D5 registre 13
07DC début écran Lo

07E0 ********************************** MC RESET PRINTER

restaurer point de branchement indirect pour imprimante.

07E0 adresse de départ
07E3 adresse objet
07E6 21 octets
07E9 (copier)
07EE Move (hl+3) vers ((hl+1)),cnt = (hl)
07F1 db 03 3 octets
07F2 dw BDF1 adresse objet
07F4 MC WAIT PRINTER

07F7 ********************************** convertir accents

La table suivante a été copiée par MC RESET PRINTER dans la RAM (adresse objet B804). Le premier octet de la table indique la longueur de la table en octets. Ensuite viennent plusieurs paires d'octets dont le premier indique chaque fois le code clavier et le second le caractère affecté de façon standard par l'électronique. Si cette table est modifiée dans la RAM, il est possible de manipuler les caractères affectés aux codes clavier de façon à produire, par exemple, un clavier français (AZERTY).

07F7 db 0A nombre d'octets

07F8 db A0 code clavier interne
07F9 db 5E caractère affecté ^

07FA db A1 code clavier interne
07FB db 5C caractère affecté \

07FC db A2 code clavier interne
07FD db 7B caractère affecté {

07FE db A3 code clavier interne
07FF db 23 caractère affecté #

0800 db A6 code clavier interne
0801 db 40 caractère affecté @

0802 db AB code clavier interne
0803 db 7C caractère affecté |

0804 db AC code clavier interne
0805 db 7D caractère affecté }

0806 db AD code clavier interne
0807 db 7E caractère affecté ~

0808 db AE code clavier interne
0809 db 5D caractère affecté]

080A db AF code clavier interne
080B db 5B caractère affecté [

080C ***************** MC AFFECTATION DE CARACTERES

C'est ici qu'est effectuée la manipulation de conversion des accents.

080C hl: adresse de départ de la nouvelle table de caractères (RAM)

0812 convertir accents (RAM)
0817 KL LDIR CONT'D

081B ************************************** MC PRINT CHAR

Sort le caractère en a sur le port Centronics. Après retour de cette routine, le carry est mis si le caractère a été transmis avec succès.

0826 accent?
0828 sauter si non
082F MC WAIT PRINTER

0835 ************************************* MC WAIT PRINTER

Envoie un caractère à l'imprimante; si celle-ci n'est pas prête, attendre une période de délai.

0838 MC BUSY PRINTER
083B MC SEND PRINTER

0844 ************************************* MC SEND PRINTER

Envoie un caractère à l'imprimante qui ne doit pas être occupée.

0847 octet sans strobe
0849 à l'imprimante
084E Strobe activé
0853 Strobe désactivé

0858 ************************************* MC BUSY PRINTER

Examiner si l'imprimante est occupée.

085A Port B
085E imprimante occupée
085F vers Carry

0863 ********************************** MC SOUND REGISTER

Fournir des données au Sound Controller. MC SOUND REGISTER est intéressant pour les amateurs de musique. Sans que vous vous torturiez l'esprit avec la transmission de données au PSG qui est relativement compliquée, il vous suffit de transmettre le numéro de registre et l'octet souhaités en les plaçant respectivement dans a et c.

0864 Port A
0866 Sound Register#
0868 Port C
086A Sound Chip
086C sur entrée
086C & strobe activé
0872 Strobe désactivé
0874 Port A
0876 données sound
0878 Port C
087D données
087F (insérer)

0883 *** Scan Keyboard

0883 Port A
0886 Sound Register 14 (Keyboard X Input)
0888 Port C
0891 Strobe activé
0893 Strobe désactivé
0896 Port A&B = Input
0898 Control
089D Port C
089F Keyboard Y Output et X Input
08A1 Port A
08A3 données (Keyboard X Input) vers accu
08AC Keyboard Y+1
08B0 traité tous les canaux Y?
08B2 non alors canal suivant
08B5 Port A Output
08B7 Control
08BA Port C

2.5.3 JUMP RESTORE (JRE)

Ce pack sert uniquement à affecter à nouveau aux adresses MAIN JUMP leurs valeurs par défaut.
PourlesFIRMJUMPS,unRST1 estplacédevant,pourlesARITHMETIC JUMPS, c'est un RST5.

Si vous pensez que vous avez modifié trop de vecteurs, tirez simplement la manette d'alarme en appelant JUMP RESTORE.
C'est également conseillé lorsque vous sortez d'un programme dans lequel vous avez généreusement offert au système d'exploitation vos propres routines.

08BD ************************************** JUMP RESTORE

08BD Main Jump Adress
08C0 pointeur sur zone vecteurs dans la RAM
08C3 b: nombre des vecteurs c: Code RST 1
08C6 copier table vecteurs
08C9 b: nombre des vecteurs c: Code RST 5
08CD sauver code RST
08CE pointeur+1 (RAM)
08CF un octet de la ROM dans la RAM
08D1 bc sur valeur avant LDI
08D2 complémenter accu
08D3 décaler bit 5 vers
08D4 bit 7
08D5 et isoler
08D7 aller chercher bits 0-6 d'octet fort de l'adresse
08D8 sauver octet fort
08D9 pointeur+1 (RAM)
08DA pointeur+1 (ROM)
08DB continuer tant que nécessaire
08DD retour du sous-programme

08DE ************************************** Main Jump Adress

```
08DE   dw 1B5C KM INITIALISE
08E0   dw 1B98 KM RESET
08E2   dw 1BBF KM WAIT CHAR
08E4   dw 1BC5 KM READ CHAR
08E6   dw 1BFA KM CHAR RETURN
08E8   dw 1C46 KM SET EXPAND
08EA   dw 1CB3 KM GET EXPAND
08EC   dw 1C04 KM EXPAND BUFFER
08EE   dw 1CDB KM WAIT KEY
08F0   dw 1CE1 KM READ KEY
08F2   dw 1E45 KM TEST KEY
08F4   dw 1D38 KM GET STATE
08F6   dw 1DE5 KM GET JOYSTICK
08F8   dw 1ED8 KM SET TRANSLATE
08FA   dw 1EC4 KM GET TRANSLATE
08FC   dw 1EDD KM SET SHIFT
08FE   dw 1EC9 KM GET SHIFT
0900   dw 1EE2 KM SET CONTROL
0902   dw 1ECE KM GET CONTROL
0904   dw 1E34 KM SET REPEAT
0906   dw 1E2F KM GET REPEAT
0908   dw 1DF6 KM SET DELAY
090A   dw 1DF2 KM GET DELAY
090C   dw 1DFA KM ARM BREAK
090E   dw 1E0B KM DISARM BREAK
0910   dw 1E19 KM BREAK EVENT

0912   dw 1074 TXT INITIALISE
0914   dw 1984 TXT RESET
0916   dw 1459 TXT VDU ENABLE
0918   dw 1452 TXT VDU DISABLE
091A   dw 13FE TXT OUTPUT
091C   dw 1335 TXT WR CHAR
091E   dw 13AC TXT RD CHAR
0920   dw 13A8 TXT SET GRAPHIC
0922   dw 1208 TXT WIN ENABLE
0924   dw 1252 TXT GET WINDOW
0926   dw 154F TXT CLEAR WINDOW
0928   dw 115A TXT SET COLUMN
```

```
092A   dw 1165 TXT SET ROW
092C   dw 1170 TXT SET CURSOR
092E   dw 117C TXT GET CURSOR
0930   dw 1286 TXT CUR ENABLE
0932   dw 1297 TXT CUR DISABLE
0934   dw 1276 TXT CUR ON
0936   dw 127E TXT CUR OFF
0938   dw 11CA TXT VALIDATE
093A   dw 1265 TXT PLACE/REMOVE CURSOR
093C   dw 1265 TXT PLACE/REMOVE CURSOR
093E   dw 12A6 TXT SET PEN
0940   dw 12BA TXT GET PEN
0942   dw 12AB TXT SET PAPER
0944   dw 12C0 TXT GET PAPER
0946   dw 12C6 TXT INVERSE
0948   dw 137B TXT SET BACK
094A   dw 1388 TXT GET BACK
094C   dw 12D4 TXT GET MATRIX
094E   dw 12F2 TXT SET MATRIX
0950   dw 12FE TXT SET M TABLE
0952   dw 132B TXT GET M TABLE
0954   dw 14D4 TXT GET CONTROLS
0956   dw 10E4 TXT STR SELECT
0958   dw 1103 TXT SWAP STREAMS

095A   dw 15A8 GRA INITIALISE
095C   dw 15D7 GRA RESET
095E   dw 15FE GRA MOVE ABSOLUTE
0960   dw 15FB GRA MOVE RELATIVE
0962   dw 1606 GRA ASK CURSOR
0964   dw 160E GRA SET ORIGIN
0966   dw 161C GRA GET ORIGIN
0968   dw 16A5 GRA WIN WIDTH
096A   dw 16EA GRA WIN HEIGHT
096C   dw 1717 GRA GET W WIDTH
096E   dw 172D GRA GET W HEIGHT
0970   dw 1736 GRA CLEAR WINDOW
0972   dw 1767 GRA SET PEN
0974   dw 1775 GRA GET PEN
```

```
0976    dw 176E GRA SET PAPER
0978    dw 177A GRA GET PAPER
097A    dw 1783 GRA PLOT ABSOLUTE
097C    dw 1780 GRA PLOT RELATIVE
097E    dw 1797 GRA TEST ABSOLUTE
0980    dw 1794 GRA TEST RELATIVE
0982    dw 17A9 GRA LINE ABSOLUTE
0984    dw 17A6 GRA LINE RELATIVE
0986    dw 1940 GRA WR CHAR

0988    dw 0ABF SCR INITIALISE
098A    dw 0AD0 SCR RESET
098C    dw 0B37 SCR SET OFFSET
098E    dw 0B3C SCR SET BASE
0990    dw 0B56 SCR GET LOCATION
0992    dw 0AE9 SCR SET MODE
0994    dw 0B0C SCR GET MODE
0996    dw 0B17 SCR MODE CLEAR
0998    dw 0B5D SCR CHAR LIMITS
099A    dw 0B6A SCR CHAR POSITION
099C    dw 0BAF SCR DOT POSITION
099E    dw 0C05 SCR NEXT BYTE
09A0    dw 0C11 SCR PREV BYTE
09A2    dw 0C1F SCR NEXT LINE
09A4    dw 0C39 SCR PREV LINE
09A6    dw 0C8E SCR INK ENCODE
09A8    dw 0CA7 SCR INK DECODE
09AA    dw 0CF2 SCR SET INK
09AC    dw 0D1A SCR GET INK
09AE    dw 0CF7 SCR SET BORDER
09B0    dw 0D1F SCR GET BORDER
09B2    dw 0CEA SCR SET FLASHING
09B4    dw 0CEE SCR GET FLASHING
09B6    dw 0DB9 SCR FILL BOX
09B8    dw 0DBD SCR FLOOD BOX
09BA    dw 0DE5 SCR CHAR INVERT
09BC    dw 0E00 SCR HW ROLL
09BE    dw 0E44 SCR SW ROLL
09C0    dw 0EF9 SCR UNPACK
```

```
09C2   dw 0F2A SCR REPACK
09C4   dw 0C55 SCR ACCESS
09C6   dw 0C74 SCR PIXELS
09C8   dw 0F93 SCR HORIZONTAL
09CA   dw 0F9B SCR VERTICAL

09CC   dw 24BC CAS INITIALISE
09CE   dw 24CE CAS SET SPEED
09D0   dw 24E1 CAS NOISY
09D2   dw 2BBB CAS START MOTOR
09D4   dw 2BBF CAS STOP MOTOR
09D6   dw 2BC1 CAS RESTORE MOTOR
09D8   dw 24E5 CAS IN OPEN
09DA   dw 2550 CAS IN CLOSE
09DC   dw 2557 CAS IN ABANDON
09DE   dw 25A0 CAS IN CHAR
09E0   dw 2618 CAS IN DIRECT
09E2   dw 2607 CAS RETURN
09E4   dw 2603 CAS TEST EOF
09E6   dw 24FE CAS OUT OPEN
09E8   dw 257F CAS OUT CLOSE
09EA   dw 2599 CAS OUT ABANDON
09EC   dw 25C6 CAS OUT CHAR
09EE   dw 2653 CAS OUT DIRECT
09F0   dw 2692 CAS CATALOG
09F2   dw 29AF CAS WRITE
09F4   dw 29A6 CAS READ
09F6   dw 29C1 CAS CHECK

09F8   dw 1FE9 SOUND RESET
09FA   dw 2114 SOUND QUEUE
09FC   dw 21CE SOUND CHECK
09FE   dw 21EB SOUND ARM EVENT
0A00   dw 21AC SOUND RELEASE
0A02   dw 2050 SOUND HOLD
0A04   dw 206B SOUND CONTINUE
0A06   dw 2495 SOUND AMPL ENVELOPE
0A08   dw 249A SOUND TONE ENVELOPE
0A0A   dw 24A6 SOUND A ADRESS
```

```
0A0C  dw 24AB SOUND T ADRESS

0A0E  dw 005C KL CHOKE OFF
0A10  dw 0326 KL ROM WALK
0A12  dw 0330 KL INIT BACK
0A14  dw 02A0 KL LOG EXT
0A16  dw 02B1 KL FIND COMMAND
0A18  dw 0163 KL NEW FRAME FLY
0A1A  dw 016A KL ADD FRAME FLY
0A1C  dw 0170 KL DEL FRAME FLY
0A1E  dw 0176 KL NEW FAST TICKER
0A20  dw 017D KL ADD FAST TICKER
0A22  dw 0183 KL DEL FAST TICKER
0A24  dw 01B3 KL ADD TICKER
0A26  dw 01C5 KL DEL TICKER
0A28  dw 01D2 KL INIT EVENT
0A2A  dw 01E2 KL EVENT
0A2C  dw 0227 KL SYNC RESET
0A2E  dw 0284 KL DELETE SYNCHRONOUS
0A30  dw 0255 KL NEXT SYNC
0A32  dw 0219 KL DO SYNC
0A34  dw 0276 KL DONE SYNC
0A36  dw 0294 KL EVENT DISABLE
0A38  dw 029A KL EVENT ENABLE
0A3A  dw 028D KL DISARM EVENT
0A3C  dw 0099 KL TIME PLEASE
0A3E  dw 00A3 KL TIME SET

0A40  dw 05ED MC BOOT PROGRAM
0A42  dw 061C MC START PROGRAM
0A44  dw 07B4 MC WAIT FLYBACK
0A46  dw 0776 MC SET MODE
0A48  dw 07C0 MC SCREEN OFFSET
0A4A  dw 0786 MC CLEAR INKS
0A4C  dw 078C MC SET INKS
0A4E  dw 07E0 MC RESET PRINTER
0A50  dw 081B MC PRINT CHAR
0A52  dw 0858 MC BUSY PRINTER
0A54  dw 0844 MC SEND PRINTER
```

0A56 dw 0863 MC SOUND REGISTER

0A58 dw 08BD JUMP RESTORE

0A5A dw 1D3C KM SET STATE
0A5C dw 1BFE KM VIDER BUFFER
0A5E dw 1460 TXT FLAG CURSEUR ACTUEL VERS ACCU
0A60 dw 15EC GRA NN
0A62 dw 19D5 GRA SAUVER PARAMETRES
0A64 dw 17B0 GRA SAUVER PARAMETRES MASQUE
0A66 dw 17AC GRA SAUVER PARAMETRES MASQUE
0A68 dw 1624 GRA CONVERTIR COORD.
0A6A dw 19D9 GRA FILL
0A6C dw 0B45 SCR MODIFIER DEBUT ECRAN
0A6E dw 080C MC AFFECTATION DE CARACTERES
0A70 dw 0397 KL FIXER CONFIGURATION RAM

0A72 ************************************ BASIC Jump Adr.

0A72 dw 2C02 EDIT

0A74 dw 2F91 FLO COPIER VARIABLE DE (DE) VERS (HL)
0A76 dw 2F9F FLO ENTIER VERS VIRGULE FLOTTANTE
0A78 dw 2FC8 FLO VALEUR 4 OCTETS VERS FLO
0A7A dw 2FD9 FLO FLO VERS ENTIER
0A7C dw 3001 FLO FLO VERS ENTIER
0A7E dw 3014 FLO FIX
0A80 dw 3055 FLO INT
0A82 dw 305F FLO
0A84 dw 30C6 FLO MULTIPLIER UN NOMBRE PAR 10^A
0A86 dw 34A2 FLO ADDITION
0A88 dw 3159 FLO RND
0A8A dw 349E FLO SOUSTRACTION
0A8C dw 3577 FLO MULTIPLICATION
0A8E dw 3604 FLO DIVISION
0A90 dw3188FLOALLERCHERCHERDERNIEREVALEURRND
0A92 dw 36DF FLO COMPARAISON
0A94 dw 3731 FLO CHANGEMENT DE SIGNE
0A96 dw 3727 FLO SGN

0A98 dw 3345 FLO DEG/RAD
0A9A dw 2F73 FLO PI
0A9C dw 32AC FLO SQR
0A9E dw 32AF FLO ELEVATION A LA PUISSANCE
0AA0 dw 31B6 FLO LOG
0AA2 dw 31B1 FLO LOG10
0AA4 dw 322F FLO EXP
0AA6 dw 3353 FLO SIN
0AA8 dw 3349 FLO COS
0AAA dw 33C8 FLO TAN
0AAC dw 33D8 FLO ATN
0AAE dw 2FD1 FLO VALEUR 4 OCTETS VERS FLO
0AB0 dw 3136 FLO RND INIT
0AB2 dw 3143 FLO SET RND SEED

0AB4 *********************** Move (hl+3) vers ((hl+1)),cnt=(hl)

2.5.4 SCREEN PACK (SCR)

Le SCREEN PACK est subordonné au TEXT PACK et au GRAPHICS PACK. Il se charge de la réalisation pratique des tâches ordonnées par ces deux packs. Il est en effet responsable du traitement direct de l'écran.

0ABF ************************************ SCR INITIALISE

initialisation complète du pack écran.

0ABF couleurs défaut
0AC2 MC CLEAR INKS
0AC7 (octet fort début écran)
0ACA SCR RESET

0AD0 ** SCR RESET

réinitialiser le pack écran.

0AD1 SCR ACCESS
0AD4 Restore SCR Indirections
0AD7 Move (hl+3) vers ((hl+1)),cnt=(hl)
0ADA Reset couleurs
0ADD db 09 9 octets
0ADE dw BDE5 adresse objet
0AE0 SCR READ
0AE3 SCR WRITE
0AE6 SCR CLEAR

0AE9 *** SCR SET MODE

mettre en place nouveau mode écran.

0AFF SCR CLEAR

0B0C *** SCR GET MODE

aller chercher mode écran actuel.

0B0C (curr. Screen Mode)

0B17 ** SCR CLEAR

vider l'écran.

0B1D SCR SET OFFSET
0B25 hl=adresse de base
0B26 de=adresse de base+1
0B28 16k
0B2C vider l'écran

0B31 (curr. Screen Mode)
0B34 MC SET MODE

0B37 ** SCR SET OFFSET

fixer adresse de départ du premier caractère relativement à l'adresse de base de la RAM vidéo.

0B37 (octet fort début écran)

0B3C *** SCR SET BASE

adresse de base de la RAM vidéo

0B3C (Position à l'intérieur d'une ligne)
0B42 MC SCREEN OFFSET

0B45 **************** SCR MODIFIER DEBUT ECRAN

0B47 (octet fort début écran)
0B51 (Position à l'intérieur d'une ligne)

0B56 *********************************** SCR GET LOCATION

début écran actuel? (base + offset)

0B56 (Position à l'intérieur d'une ligne)
0B59 (octet fort début écran)

0B5D ************************************ SCR CHAR LIMITS

Aller chercher nombres maxi de lignes et colonnes de l'écran (en fonction du mode).

0B5D SCR GET MODE

0B6A *********************************** SCR CHAR POSTION

traduire coordonnées physiques en une position écran

0B6B SCR GET MODE
0B93 (octet fort début écran)
0BA6 SCR CHAR POSITION

0BAF ************************************ SCR DOT POSITION

déterminer position écran pour un pixel.

0BED (octet fort début écran)

0BF6 SCR GET MODE

0C05 ** SCR NEXT BYTE

SCRNEXTBYTEetSCRPREVBYTEfournissentdanshl l'adresseécran de la prochaine ou de la dernière position d'octet, lorsque vous placez dans hl, avant d'appeler la routine, l'ancienne adresse. C'est aussi pratique que cela semble superflu. En effet, du fait de l'organisation de l'écran, il n'est pas facile de déterminer la position d'octet. La distance dépend en outre du mode. Notez que si la prochaine ou la dernière position sort du cadre de l'écran, l'adresse fournie en retour n'a pas de sens. Elles se trouve en effet alors dans la zone des 48 derniers octets de la Ram vidéo, qui ne sont pas utilisés pour la représentation sur l'écran.

0C11 ** SCR PREV BYTE

Voir SCR NEXT BYTE.

0C1F ** SCR NEXT LINE

SCR NEXT LINE et SCR PREV LINE travaillent de façon similaire à SCR NEXT BYTE, si ce n'est que l'adresse écran est calculée une ligne entière avant ou aprés.
Ici également, l'adresse n'a pas de signification lorsqu'on sort de la zone représentable.

0C39 ** SCR PREV LINE

Voir SCR NEXT LINE

0C55 *** SCR ACCESS

fixer caractères de commande sur visible/invisible.

0C57 SCR PIXELS (FORCE MODE)
0C5E Low Byte XOR Mode
0C62 Low Byte AND Mode
0C66 Low Byte OR Mode
0C68 jp
0C6A (Write Indirection)

0C71 *** SCR WRITE

0C71 Write Indirection

0C74 **************************** SCR PIXELS (FORCE Mode)

Fixer point sur l'écran.

0C7A *** XOR Mode

0C7F *** AND Mode

0C85 *** OR Mode

0C8A *** SCR READ

0C8E ************************************* SCR INK ENCODE

codage d'une ink de façon à ce que tous les points image soient fixés sur cette ink.

0CA7 ************************************* SCR INK DECODE

décodage d'une ink.

0CC9 SCR GET MODE

0CD8 *** Reset couleurs

0CD8 couleurs défaut
0CDB mémoire couleur 2des couleurs
0CE4 (flag jeu de couleurs actuel)

0CEA ************************************* SCR SET FLASHING

fixer durées de clignotement des couleurs pour toutes les inks et pour le bord.

0CEA (Flash Periods)

0CEE ************************************* SCR GET FLASHING

déterminer durées de clignotement (inks et bord).

0CEE (Flash Periods)

0CF2 ** SCR SET INK

affectation des deux couleurs utilisées pour représenter une ink.

0CF5 Set Colour

0CF7 ** SCR SET BORDER

affectation des deux couleurs utilisées pour représenter un bord

0CF8 ** Set Colour

0CFA aller chercher entrée matrice couleur
0CFF aller chercher entrée matrice couleur
0D04 aller chercher adresse ink

0D10 *********************************** aller chercher entrée matrice couleur

0D1A *************************************** SCR GET INK

aller chercher les deux couleurs utilisées pour représenter une ink.

0D1D Get Colour

0D1F ************************************ SCR GET BORDER

aller chercher les deux couleurs utilisées pour représenter un cadre.

0D20 *** Get Colour

0D20 aller chercher adresse ink
0D2C matrice couleurs

0D35***************************************allercheradresse ink

0D38 mémoire couleurs 1ères couleurs
0D42 Event Block: Set Inks
0D46 KL DEL FRAME FLY
0D49 Flash Inks
0D4C Set Inks on Frame Fly
0D52 KL NEW FRAME FLY
0D55 Event Block: Set Inks
0D58 KL DEL FRAME FLY
0D5B aller chercher paramètres du jeu de couleurs actuel
0D5E MC CLEAR INKS

0D61 *********************************** Set Inks on Frame Fly

0D61 curr. Flash Period
0D65 Flash Inks
0D6B aller chercher paramètres du jeu de couleurs actuel
0D6E MC SET INKS

0D73 *** Flash Inks

0D73 aller chercher paramètres du jeu de couleurs actuel
0D76 (curr. Flash Period)
0D79 MC SET INKS
0D7C flag jeu de couleurs actuel

0D87 ***************************amener param. du jeu de couleurs actuel

0D87 mémoire couleurs 1ères couleurs
0D8A (flag jeu de couleurs actuel)
0D8E (Flash Period 1. Colour)
0D92 mémoire couleurs 2èmes couleurs
0D95 (Flash Periods)

0D99***matricecouleurs

0D99 14 04 15 1C 18 1D 0C 05
0DA1 0D 16 06 17 1E 00 1F 0E
0DA9 07 0F 12 02 13 1A 19 1B
0DB1 0A 03 0B 01 08 09 10 11

0DB9 *** SCR FILL BOX

Remplir fenêtre indiquée avec une couleur (positions en caractères, en fonction du mode).

0DBD ************************************** SCR FLOOD BOX

remplir fenêtre indiquée avec une couleur (les positions sont des adresses écran, indépendantes du mode).

0DC6 SCR NEXT BYTE
0DDE SCR NEXT LINE
0DE2 SCR FLOOD BOX

0DE5 ************************************ SCR CHAR INVERT

echanger les couleurs de premier et second plans d'un caractère.

0DE8   SCR CHAR POSITION
0DF2   SCR NEXT BYTE

0DF8 ******************************** adresser mémoire couleurs

0DF9   SCR NEXT LINE

0E00 *** SCR HW ROLL

SCR HW ROLL décale l'écran (en hardware) d'une ligne vers le bas lorsque b=0 et d'une ligne vers le haut lorsque b<>0.
a doit recevoir la couleur que devra avoir la nouvelle ligne (vide) qui sera ajoutée.

0E0B   MC WAIT FLYBACK
0E32   (octet fort début écran)
0E3A   SCR FLOOD BOX
0E41   SCR SET OFFSET

0E44 *** SCR SW ROLL

SCR SW ROLL décale une zone de l'écran (décalage software). a et b doivent être servis comme ci-dessus. h doit en outre recevoir le numéro de colonne du bord gauche de la zone à décaler, l la ligne supérieure, d la colonne droite et e la ligne inférieure de cette zone.
Notez que colonne et ligne 0 correspondent à l'angle supérieur gauche de l'écran. Faites vous-même très attention à ce que les paramètres transmis marquent bien une zone comprise dans la Ram vidéo.

0E4F   SCR CHAR POSITION
0E5A   MC WAIT FLYBACK
0E64   SCR NEXT LINE
0E69   SCR NEXT LINE
0E76   SCR FLOOD BOX
0E8B   SCR CHAR POSITION
0E8F   SCR CHAR POSITION
0E93   MC WAIT FLYBACK

0E96 SCR PREV LINE
0E9B SCR PREV LINE
0EE1 SCR NEXT BYTE
0EE5 SCR NEXT BYTE

0EF9 *** SCR UNPACK

agrandir matrice caractère (pour modes 0/1).

0EF9 SCR GET MODE

0F2A *** SCR REPACK

rétablir matrice caractère dans sa forme originelle.

0F2B SCR CHAR POSITION
0F2E SCR GET MODE
0F3C SCR NEXT LINE
0F48 SCR NEXT BYTE
0F53 SCR NEXT LINE
0F82 SCR NEXT BYTE
0F8C SCR NEXT LINE

0F93 ************************************ SCR HORIZONTAL

tracer ligne horizontale.

0F9B ** SCR VERTICAL

tracer ligne verticale.

0FA5 (GRA Pen)
0FA9 (GRA Pen)
0FAE (GRA Pen)
0FB1 (GRA Pen)
0FB8 charger &FF dans accu
0FF3 (GRA Paper)
0FFF (GRA Pen)
100A SCR NEXT BYTE

101C (GRA Pen)
1027 (GRA Paper)
102C SCR WRITE
1030 SCR PREV LINE
1049 SCR DOT POSITION

1052 ** couleurs défaut

1052 04 04 0A 13 0C 0B 14 15
105A 0D 06 1E 1F 07 12 19 04
1062 17 04 04 0A 13 0C 0B 14
106A 15 0D 06 1E 1F 07 12 19
1072 0A 07

2.5.5 TEXT SCREEN (TXT)

Ce pack est responsable de la gestion de textes, ce qui comprend également l'organisation des fenêtres.

Quelques remarques sont nécessaires en ce qui concerne la manipulation du curseur:

Les coordonnées réclamées ou fournies par les routines du curseur doivent être comprises comme des indications logiques, c'est-à-dire qu'elles se rapportent à la fenêtre actuelle. Les coordonnées 1,1 correspondent à l'angle supérieur gauche de la fenêtre. Si vous voulez par exemple positionner, avec TXT SET CURSOR, le curseur en dehors de la fenêtre, il sera automatiquement fixé sur la prochaine position possible à l'intérieur de la fenêtre, si le curseur est activé ou si un caractère doit être représenté ensuite.

La position actuelle (que vous pouvez lire avec TXT GET CURSOR) est ainsi également modifiée.
Si le curseur est désactivé, la nouvelle position souhaitée est d'abord acceptée, jusqu'à ce qu'un caractère soit représenté ou jusqu'à ce que le curseur soit activé.

1074 ** TXT INITIALISE

initialisation complète du pack texte.

1074 TXT RESET
107E TXT fixer paramètres défaut
1081 Reset Params (toutes les fenêtres)

1084 *** TXT RESET

réinitialiser le pack texte.

1084 Restore TXT Indirections
1087 Move (hl+3) vers ((hl+1)), cnt=(hl)
108D db 0F 15 octets
108E dw BDCD adresse objet
1090 TXT DRAW/UNDRAW CURSOR
1093 TXT DRAW/UNDRAW CURSOR
1096 TXT WRITE CHAR
1099 TXT UNWRITE CHAR
109C TXT OUT ACTION

109F ****************************** Reset Params (toutes les fenêtres)

10A1 début paramètres fenêtre 0
10A4 position curseur actuelle (Row, Col)
10AF (fenêtre écran actuelle)
10B3 (fenêtre écran actuelle)
10BB TXT STR SELECT
10BE TXT DRAW/UNDRAW CURSOR
10C1 TXT GET PAPER
10C4 (TXT paper actuel)
10C7 TXT GET PEN
10CA (TXT pen actuel)
10D6 TXT STR SELECT
10DA (TXT pen actuel)
10DD fixer paramètres défaut

10E4 *************************************** TXT STR SELECT

sélectionner fenêtre de texte

10E6 fenêtre écran actuelle
10F1 Adr. paramètres fenêtre vers de
10F4 ldir cnt=15
10F8 Adr. paramètres fenêtre vers de
10FC ldir cnt=15

1103 ************************************ TXT SWAP STREAMS

échanger les paramètres (couleurs, limites de fenêtre etc.) de deux fenêtres.

1103 (fenêtre écran actuelle)
1108 TXT STR SELECT
110C (fenêtre écran actuelle)
110F Adr. paramètres fenêtre vers de
1114 Adr. paramètres fenêtre vers de
1118 ldir cnt=15
111C TXT STR SELECT

111E ** ldir cnt=15

1126 ****************************** Adr. paramètres fenêtre vers de

1135 position curseur actuelle (Row, Col)

1139 ************************************ fixer paramètres défaut

113C (flag curseur actuel)
1140 TXT SET PAPER
1144 TXT SET PEN
1148 TXT SET GRAPHIC
114B TXT SET BACK
1154 TXT WIN ENABLE
1157 TXT VDU ENABLE

115A ************************************ TXT SET COLUMN

fixer position horizontale du curseur.

115B fenêtre actuelle gauche
115F (position curseur actuelle (Row, Col))

1165 ** TXT SET ROW

fixer position verticale du curseur.

1166	fenêtre actuelle haut
116A	(position curseur actuelle (Row, Col)

1170 *********************************** TXT SET CURSOR

positionner le curseur.

1170	fenêtre actuelle haut, gauche + hl
1173	TXT DRAW/UNDRAW CURSOR
1176	(position curseur actuelle (Row, Col)
1179	TXT DRAW/UNDRAW CURSOR

117C *********************************** TXT GET CURSOR

demander la position actuelle du curseur.

117C	(position curseur actuelle (Row, Col))
117F	fenêtre actuelle haut, gauche - hl
1182	(act. Roll Count)

1186 ****************************** fenêtre actuelle haut, gauche + hl

1186	(fenêtre actuelle haut)
118C	(fenêtre actuelle gauche)

1193 ****************************** fenêtre actuelle haut, gauche - hl

1193	(fenêtre actuelle haut)
119B	(fenêtre actuelle gauche)

11A4 *** Move Cursor

11A4	TXT DRAW/UNDRAW CURSOR
11A7	(position curseur actuelle (Row, Col))
11AA	hl à l'intérieur limites fenêtre?
11AD	(position curseur actuelle (Row, Col))
11B2	act. Roll Count

11BA TXT GET WINDOW
11BD (TXT paper actuel)
11C1 SCR SW ROLL
11C5 SCR HW ROLL

11CA ************************************* TXT VALIDATE

curseur à l'intérieur de la fenêtre de texte?

11CA fenêtre actuelle haut, gauche + hl
11CD hl à l'intérieur limites fenêtre?
11D1 fenêtre actuelle haut, gauche - hl

11D6 ****************************** hl à l'intérieur limites fenêtre

11D6 (fenêtre actuelle droite)
11DD (fenêtre actuelle gauche)
11E2 (fenêtre actuelle gauche)
11E7 (fenêtre actuelle droite)
11EF (fenêtre actuelle haut)
11F7 (fenêtre actuelle bas)

1208 ************************************ TXT WIN ENABLE

déterminer taille de la fenêtre de texte actuelle.

1208 SCR CHAR LIMITS
1229 (fenêtre actuelle haut)
122C (fenêtre actuelle bas)
123A (flag fenêtre (0=écran activé))

1252 ************************************ TXT GET WINDOW

Quelle taille a la fenêtre de texte actuelle?

1252 (fenêtre actuelle haut)
1255 (fenêtre actuelle bas)
1259 (flag fenêtre (0=écran activé))

125F *********************** TXT DRAW/UNDRAW CURSOR

fixer/supprimer le curseur.

125F   (flag curseur actuel)

1265 *********************** TXT PLACE/REMOVE CURSOR

fixer curseur sur l'écran/enlever curseur de l'écran.

126B   (TXT pen actuel)
126F   SCR CHAR INVERT

1276 *** TXT CUR ON

autoriser curseur (système d'exploitation).

1279   Cur Enable Cont'd

127E *** TXT CUR OFF

verrouiller curseur (système d'exploitation, priorité supérieure à TXT CUR ENABLE et TXT CUR DISABLE.

1281   Cur Disable Cont'd

1286 ************************************* TXT CUR ENABLE

autoriser curseur (programme utilisateur).

1288 *************************************** Cur Enable Cont'd

1289   TXT DRAW/UNDRAW CURSOR
128E   flag curseur actuel
1294   TXT DRAW/UNDRAW CURSOR

1297 ************************************* TXT CUR DISABLE

verrouiller curseur (programme utilisateur).

1299 ************************************* Cur Disable Cont'd

129A TXT DRAW/UNDRAW CURSOR
129F flag curseur actuel

12A6 ** TXT SET PEN

fixer couleur de premier plan.

12A6 TXT pen actuel

12AB ************************************* TXT SET PAPER

fixer couleur d'arrière-plan

12AB TXT act. Paper
12AF TXT DRAW/UNDRAW CURSOR
12B3 SCR INK ENCODE
12B7 TXT DRAW/UNDRAW CURSOR

12BA *************************************** TXT GET PEN

quelle couleur de premier plan est-elle mise?

12BA (TXT pen actuel)
12BD SCR INK DECODE

12C0 ************************************* TXT GET PAPER

quelle couleur d'arrière plan est-elle mise?

12C0 (TXT paper actuel)
12C3 SCR INK DECODE

12C6 ** TXT INVERSE

échanger couleurs de premier et arrière plans actuelles.

12C6 TXT DRAW/UNDRAW CURSOR

12C9 (TXT pen actuel)
12CF (TXT pen actuel)

12D4 ************************************ TXT GET MATRIX

aller chercher adresse du modèle en points d'un caractère.

12D6 TXT GET M TABLE

12F2 ************************************* TXT SET MATRIX

aller chercher adresse du modèle points (défini par l'utilisateur) d'un caractère déterminé.

12F3 TXT GET MATRIX

12FE ************************************ TXT SET M TABLE

fixer adresse de départ et premier caractère d'un matrice de points définie par l'utilisateur.

130A TXT GET MATRIX
131E TXT GET M TABLE
1321 (1er caractère User Matrix)
1326 (Adr. User Matrix)

132B *********************************** TXT GET M TABLE

adresse de départ et premier caractère d'une matrice utilisateur?

132B (1er caractère User Matrix)
1331 (Adr. User Matrix)

1335 *** TXT WR CHAR

représenter caractère.

1336 (flag curseur actuel)
133C move Cursor

1340 (position curseur actuelle (Row, Col))
1345 TXT WRITE CHAR
1348 TXT DRAW/UNDRAW CURSOR

134B ************************************ TXT WRITE CHAR

écrire un caractère sur l'écran.

134C TXT GET MATRIX
1353 SCR UNPACK
1358 SCR CHAR POSITION
1366 SCR NEXT BYTE
136F SCR NEXT LINE
1377 (act. mode fond)

137B ** TXT SET BACK

mode transparent activé/désactivé.

1384 (act. mode fond)

1388 ** TXT GET BACK

quel mode transparent?

1388 (act. mode fond)
1392 (TXT pen actuel)
13A0 (TXT pen actuel)
13A5 SCR PIXELS

13A8 ************************************ TXT SET GRAPHIC

activer ou désactiver la représentation de caractères de commande.

13A8 (GRA Char WR Mode (0=disable))

13AC ** TXT RD CHAR

lire un caractère de l'écran.

13AF move Cursor
13B2 TXT UNWRITE CHAR
13B6 TXT DRAW/UNDRAW CURSOR

13BE ********************************* TXT UNWRITE CHAR

lire un caractère de l'écran.

13BE (TXT pen actuel)
13C6 SCR REPACK
13DE SCR REPACK
13E4 TXT GET MATRIX

13FE *** TXT OUTPUT

représenter ou exécuter caractères (de commande).

Amène le caractère dans l'accumulateur sur la fenêtre écran actuelle ou bien l'exécute s'il s'agit d'un caractère de commande. Notez que cette routine utilise l'indirection TXT OUT ACTION. Si vous l'avez 'détournée' TXT OUTPUT utilisera aussi votre routine et non la routine de la ROM.

1402 TXT OUT ACTION

140A ************************************ TXT OUT ACTION

Sortie d'un caractère sur l'écran ou exécution d'un code de commande.

140B (GRA Char WR Mode (0=disable)
1410 GRA WR CHAR
1413 compteur de caractères Control Buffer
1418 Control Buffer plein?
141A oui, alors sauter
141C Control Buffer vide?
141D non, alors sauter
1420 caractère de commande?
1422 non, alors TXT WR CHAR

1425 compteur+1
142C (Start Control Buffer)
1430 table de saut caractère de commande
1436 nombre requis
1439 atteint paramètres de commande
143A non, alors sauter
1446 Start Control Buffer
144A call (de)
144E (compteur de caractères Control Buffer)

1452 ************************************ TXT VDU DISABLE

inhiber représentation du caractère.

1454 Cur Disable Cont'd

1459 ************************************ TXT VDU ENABLE

On peut écrire des caractères sur l'écran.

145B Cur Enable Cont'd

1460******************* FLAG CURSEUR ACTUEL VERS ACCU

1460 (flag curseur actuel)

1464 ****************copier sauts caractères de commande défaut

1465 (compteur de caractères Control Buffer)
1468 sauts caractères de commande défaut
146B table de saut caractère de commande
146E nombre d'octets
1471 copier

1474 **************************** sauts caractères de commande défaut

1474 db 80
1475 dw 1513 00

```
1477    db 81
1478    dw 1335  01 TXT WR CHAR

147A    db 80
147B    dw 1297  02 TXT CUR DISABLE

147D    db 80
147E    dw 1286  03 TXT CUR ENABLE

1480    db 81
1481    dw 0AE9  04 SCR SET MODE

1483    db 81
1484    dw 1940  05 GRA WR CHAR

1486    db 00
1487    dw 1459  06 TXT VDU ENABLE

1489    db 80
148A    dw 14E1  07 bip-bip

148C    db 80
148D    dw 1519  08 CRSR Left

148F    dw 80
1490    dw 151E  09 CRSR Right

1492    db 80
1493    dw 1523  0A CRSR Down

1495    db 80
1496    dw 1528  0B CRSR Up

1498    db 80
1499    dw 154F  0C TXT CLEAR WINDOW

149B    db 80
149C    dw 153F  0D CRSR sur début de ligne
```

```
149E   db 81
149F   dw 12AB  0E TXT SET PAPER

14A1   db 81
14A2   dw 12A6  0F TXT SET PEN

14A4   db 80
14A5   dw 155E  10 supprimer caractère dans position CRSR

14A7   db 80
14A8   dw 1599  11 supprimer ligne jusqu'à position CRSR

14AA   db 80
14AB   dw 158F  12 supprimer ligne à partir de position CRSR

14AD   db 80
14AE   dw 1578  13 supprimer fenêtre jusqu'à position CRSR

14B0   db 80
14B1   dw 1565  14 supprimer fenêtre à partir de position CRSR

14B3   db 80
14B4   dw 1452  15 TXT VDU DISABLE

14B6   db 81
14B7   dw 14EC  16 mode transparent activé/désactivé

14B9   db 81
14BA   dw 0C55  17 SCR ACCESS

14BC   db 80
14BD   dw 12C6  18 TXT INVERSE

14BF   db 89
14C0   dw 150D  19 instruction SYMBOL

14C2   db 84
14C3   dw 1501  1A définir fenêtre
```

14C5 db 00
14C6 dw 14EB 1B aucun effet

14C8 db 83
14C9 dw 14F1 1C instruction INK

14CB db 82
14CC dw 14FA 1D instruction BORDER

14CE db 80
14CF dw 1539 1E CRSR Home

14D1 db 82
14D2 dw 1547 1F instruction LOCATE

14D4 ********************************* TXT GET CONTROLS

aller chercher adresse de la table de saut caractères de commande.

14D4 table de saut caractère de commande

14E1 ** bip-bip

14E6 SOUND QUEUE

14EC ******************************* mode transparent activé/désactivé

14EE TXT SET BACK

14F1***instructionINK

14F7 SCR SET INK

14FA***************************************instructionBORDER

14FE SCR SET BORDER

1501 ************************************** définir fenêtre

150A TXT WIN ENABLE

150D ******************************* instruction SYMBOL

1510 TXT SET MATRIX

1513 Move Cursor
1516 TXT DRAW/UNDRAW CURSOR

1519 ** CRSR Left

151E ** CRSR Right

1523 ** CRSR Down

1528 ** CRSR Up

152C Move Cursor

1539 ** CRSR Home

153F ********************************* CRSR sur début de ligne

153F Move Cursor
1542 (fenêtre actuelle gauche)

1547**instructionLOCATE

154C TXT SET CURSOR

154F ********************************* TXT CLEAR WINDOW

vider fenêtre de texte actuelle.

154F TXT DRAW/UNDRAW CURSOR

1552 (fenêtre actuelle haut)
1555 (position curseur actuelle (Row, Col)
1558 (fenêtre actuelle bas)

155E ************************* supprimer caractère dans position CRSR

155E Move Cursor

1565 **************************** supprimer fenêtre à partir de position CRSR

1565 12 supprimer lignes à partir de position CRSR
1568 (fenêtre actuelle haut)
156B (fenêtre actuelle bas)
156F (position curseur actuelle (Row, Col))

1578 ************************** supprimer fenêtre jusqu'à position CRSR

1578 11 supprimer lignes jusqu'à position CRSR
157B (fenêtre actuelle haut)
157E (fenêtre actuelle droite)
1582 (position curseur actuelle (Row, Col)
1589 (TXT paper actuel)
158C SCR FILL BOX

158F ****************************** supprimer ligne à partir de position CRSR

158F Move Cursor
1593 (fenêtre actuelle droite)

1599 ***************************** supprimer ligne jusqu'à position CRSR

1599 Move Cursor
159E (fenêtre actuelle gauche)
15A5 TXT DRAW/UNDRAW CURSOR

2.5.6 GRAPHICS SCREEN (GRA)

Ce pack sert exclusivement à la manipulation de la fenêtre graphique.

Au sujet des indications de coordonnées qui sont réclamées par les différentes routines, il convient de faire les remarques suivantes:

Les coordonnées sont transmises en 3 étapes. L'étape la plus proche de l'utilisateur est la position relativement à l'origine des coordonnées (ORIGIN) qu'il a lui-même fixée. Cette position est convertie en une position relativement à l'origine de l'écran (bas gauche).

Ces deux étapes sont indépendantes du mode!
La dernière étape est l'adresse physique du point. Celle-ci dépend du mode actuel!
Une étape supplémentaire peut éventuellement être ajoutée auparavant, lorsqu'une paire de coordonnées relatives doit être convertie en une position absolue relativement à ORIGIN.

15A8 *************************************** GRA INITIALISE

initialisation complète du pack graphique.

15A8 GRA RESET
15AB Pen 1,Paper 0
15AF GRA SET PAPER
15B3 GRA SET PAPER
15B6 fixer origine sur 0,0
15BB GRA SET ORIGIN
15C6 GRA WIN WIDTH
15CB GRA WIN HEIGHT
15CE GRA GET PAPER
15D2 GRA GET PEN

15D7 ** GRA RESET

réinitialisation du pack graphique.

15DA Restore GRA Indirections
15DD Move (hl+3) vers ((hl+1)), cnt=(hl)
15E0 db 09 9 octets
15E1 dw BDDC adresse objet
15E3 GRA PLOT
15E6 GRA TEST
15E9 GRA LINE

15EC ** NN

15ED SCR ACCESS
15F1 GRA FILL

15FB ******************************** GRA MOVE RELATIVE

Déplacement relativement à la position actuelle.

15FB ajouter coord. act. + coord. rel..

15FE ******************************** GRA MOVE ABSOLUTE

déplacement vers une position absolue.

15FE (act. coord. X)
1602 (act. Y Koord.)

1606 ************************************* GRA ASK CURSOR

Où est le curseur graphique actuel?

1606 (act. coord. X)
160A (act. Y Koord.)

160E ************************************** GRA SET ORIGIN

fixer origine des coordonnées utilisateur.

160E (origine X)
1612 (origine Y)
161A GRA MOVE ABSOLUTE

161C ************************************** GRA GET ORIGIN

aller chercher origine des coordonnées utilisateur.

161C (origine X)
1620 (origine Y)

1624 ********************************** aller chercher position de départ physique

1624 GRA ASK CURSOR

1627 *********************aller chercher position objet physique + fixer curseur

1627 GRA MOVE ABSOLUTE

162A ********************** GRA CONVERTIR COORD.

162B SCR GET MODE
1640 (origine X)
1655 (origine Y)

165D ***************************** ajouter coord. act. + coord. rel..

165E (act. coord. X)
1664 (act. coord. Y)
166A (coord. X GRA fenêtre gauche)
1673 (coord. X GRA fenêtre droite)
1680 (coord. Y GRA fenêtre haut)
1689 (coord. Y GRA fenêtre bas)
1694 aller chercher position objet physique + fixer curseur

16A5 ************************************* GRA WIN WIDTH

fixer limites gauche et droite de la fenêtre graphique.

16BE SCR Get Mode
16C9 (coord. X GRA fenêtre gauche)
16CD (coord. X GRA fenêtre droite)

16EA ************************************* GRA WIN HEIGHT

fixer limites supérieure et inférieure de la fenêtre graphique.

16FB (coord. Y GRA fenêtre haut)
16FF (coord. Y GRA fenêtre bas)

1717 ************************************ GRA GET W WIDTH

limites gauche et droite de la fenêtre graphique?

1717 (coord. X GRA fenêtre gauche)
171B (coord. X GRA fenêtre droite)
171E SCR GET MODE

172D ********************************** GRA GET W HEIGHT

limites supérieure et inférieure de la fenêtre graphique?

172D (coord. Y GRA fenêtre haut)
1731 (coord. Y GRA fenêtre bas)

1736 ********************************** GRA CLEAR WINDOW

vider fenêtre graphique.

1736 GRA GET W WIDTH
1746 (coord. Y GRA fenêtre bas)
174A (coord. Y GRA fenêtre haut)
1753 (coord. X GRA fenêtre gauche)
1759 SCR DOT POSITION

175D (GRA Paper)
1761 SCR FLOOD BOX

1767 ** GRA SET PEN

fixer couleur d'écriture.

1767 SCR INK ENCODE
176A (GRA Pen)

176E ** GRA SET PAPER

fixer couleur d'arrière-plan.

176E SCR INK ENCODE
1771 (GRA Paper)

1775 ** GRA GET PEN

quelle couleur d'écriture?

1775 (GRA Pen)

177A *************************************** GRA GET PAPER

quelle couleur de fond?

177A (GRA Paper)
177D (SCR INK DECODE)

1780 ********************************** GRA PLOT RELATIVE

fixer un point graphique relativement à la position actuelle du curseur.

1780 ajouter coord. act. + coord. rel.

1783 ********************************** GRA PLOT ABSOLUTE

fixer un point graphique (absolu).

1783 GRA PLOT

1786 ** GRA PLOT

représenter un point sur l'écran.

178A SCR DOR POSITION
178D (GRA Pen)
1791 SCR WRITE

1794 ********************************** GRA TEST RELATIVE

point fixé (relativement au curseur actuel)?

1794 ajouter. coord. act. + coord. rel.

1797 ********************************** GRA TEST ABSOLUTE

point fixé (absolu)?

1797 GRA TEST

179A ** GRA TEST

fournit l'ink de la position graphique actuelle.

179D GRA GET PAPER
17A0 SCR DOT POSITION
17A3 SCR READ

17A6 ********************************** GRA LINE RELATIVE

tracer une ligne de distance act. à distance relative.

17A6 ajouter. coord. act. + coord. rel.

17A9 ******************************** GRA LINE ABSOLUTE

tracer une ligne de position act. à position absolue.

17A9 GRA LINE

17AC****************** GRA SAUVER PARAMETRES MASQUE

sauver paramètres de l'instruction BASIC MASK.

17B0 ****************** GRA SAUVER PARAMETRES MASQUE

sauver paramètres de l'instruction BASIC MASK.

17B4 ** GRA LINE

dessiner une ligne.

17B9 aller chercher position objet physique + fixer curseur
17BD (buffer de calcul coord. X)
17CC (buffer de calcul coord. Y)
188C aller chercher position de départ physique
188F (buffer de calcul coord. X)
1893 (buffer de calcul coord. Y)
18A2 (buffer de calcul coord. Y)
18AD (buffer de calcul coord. Y)
18B2 (buffer de calcul coord. Y)
18B9 (coord. Y GRA fenêtre haut)
18C3 (coord. Y GRA fenêtre bas)
18C8 (buffer de calcul coord. X)
18DA (buffer de calcul coord. X)
18E6 (buffer de calcul coord. X)
18EF (buffer de calcul coord. X)
18FA (buffer de calcul coord. X)
18FF (buffer de calcul coord. X)
1906 (coord. X GRA fenêtre droite)
1910 (coord. X GRA fenêtre gauche)
1915 (buffer de calcul coord. Y)
1928 (buffer de calcul coord. Y)
1934 (buffer de calcul coord. Y)

1940 ** GRA WR CHAR

écrire un caractère dans la position curseur graphique actuelle.

1942 TXT GET MATRIX
1948 aller chercher position de départ physique
1962 SCR DOR POSITION
1973 SCR NEXT BYTE
197B SCR NEXT LINE
1985 GRA ASK CURSOR
1989 SCR GET MODE
1998 GRA MOVE ABSOLUTE
19AC SCR DOT POSITION
19C4 (GRA Pen)
19CE (GRA Paper)
19D2 SCR WRITE

19D5*******************************GRASAUVERPARAMETRES

19D9 ** GRA FILL

19D9 (buffer de calcul coord. X)
19DF (buffer de calcul coord. Y)
19E3 SCR INK ENCODE
19E9 aller chercher position de départ physique
1A19 (buffer de calcul coord. X)
1A25 (buffer de calcul coord. Y)
1A2C (buffer de calcul coord. Y)
1A44 (buffer de calcul coord. X)
1A9F (buffer de calcul coord. Y)
1AA9 (buffer de calcul coord. Y)
1AC1 (buffer de calcul coord. X)
1AE8 (GRA coord. Y GRA fenêtre haut)
1B10 SCR PREV LINE
1B18 (coord. Y GRA fenêtre bas)
1B25 SCR NEXT LINE
1B35 (GRA Pen)
1B45 SCR DOT POSITION

1B51   SCR DOT POSITION
1B56   SCR DOT POSITION

2.5.7 KEYBOARD MANAGER (KM)

Ce pack a pour fonction la surveillance du clavier et la conversion en codes de caractères utilisables.
Pour l'interrogation cyclique des touches, il utilise le mécanisme d'EVENT.

1B5C *** KM INITIALISE

Initialisation complète de la gestion clavier. L'état de la gestion clavier avant appel de cette routine est perdu.

1B5F   KM SET DELAY
1B68   (Shift Lock State)
1B80   Key Translation Table
1B8A   Key State Map
1B8D   touches enfoncées pendant examen

1B98 ** KM RESET

La gestion clavier est placé dans son état de départ. La table de saut indirect et les buffers de la gestion clavier sont neutralisés.

1BA4   Exp Buffer Cont'd
1BA7   Restore KM Indirection
1BAA   Move (hl+3) vers ((hl+1)), cnt=(hl)
1BB0   KM DISARM BREAK
1BB3   db 03   3 octets
1BB4   dw BDEE   adresse objet
1BB6   Test Break

1BBF ************************************** KM WAIT CHAR

KM WAIT CHAR va chercher un caractère dans le buffer clavier, dans la chaîne d'extension ou dans le buffer Put Back. Si aucun caractère n'est disponible, la routine ne revient pas. Elle attend obligatoirement.
a contient s'il y a lieu le caractère qui a été entré au clavier.

1BBF KM READ CHAR
1BC2 KM WAIT CHAR

1BC5 ********************************** KM READ CHAR

KM READ CHAR transmet également un caractère dans a, s'il y en avait un, mais cette routine n'attend pas qu'il y ait un résultat positif.
Si au retour de la routine, le carry est mis, c'est qu'il n'y avait pas de caractère à aller chercher.

1BC6 Put Back Buffer
1BC9 aller chercher caractère
1BCA vider buffer
1BCC y avait-il un caractère?
1BCD si oui sauter
1BCF (Exp. String Pointer)
1BD2 octet fort vers accu
1BD3 existe-t-il une chaîne d'extension?
1BD4 si oui sauter
1BD6 KM READ KEY
1BD9 sauter si aucun caractère
1BDB le caractère est-il < 128?
1BDD si < 128 alors sauter
1BE8 KM GET EXPAND
1BF0 (Exp. String Pointer)

1BF8 accu=&FF

1BFA ******************************** KM CHAR RETURN

Placer un caractère dans le buffer clavier pour le prochain accès (KM READ CHAR ou KM WAIT CHAR).

1BFA (Put Back Buffer)

1BFE KM READ CHAR

1C04 ************************************* KM EXP BUFFER

Affecter mémoire pour chaîne d'extension (adresse, longueur). Initialiser buffer

1C04 Exp Buffer Cont'd

1C0A ************************************* Exp Buffer Cont'd

1C13 (Pointer fin Exp Buffer)
1C17 (Pointer Start Exp Buffer)
1C1A ASCII
1C1D 0
1C1F jusqu'à
1C20 9
1C21 vers
1C22 Expansion
1C23 Buffer
1C25 Restore
1C26 Default Exp String
1C35 (pointeur buffer d'extension libre)

1C3C ************************************* Default Exp String

1C3C 01 2E 01 0D 05 52 55 4E .....RUN
1C44 22 0D

1C46 ************************************* KM SET EXPAND

créer chaîne d'extension.

1C47 adresse Exp String vers de
1C4A sauter si Token incorrect
1C4E vider buffer d'extension

1C6A *********************************** vider buffer d'extension

1C79 place pour une nouvelle chaîne d'extension?
1C85 (pointeur buffer d'extension libre)
1C8A (Pointer fin Exp Buffer)
1C93 place pour une nouvelle chaîne d'extension?
1C96 (pointeur buffer d'extension libre)
1CA1 (pointeur buffer d'extension libre)

1CA7 ***************************** place pour une nouvelle chaîne d'extension?

1CA7 (pointeur buffer d'extension libre)

1CB3 ************************************* KM GET EXPAND

Aller chercher un caractère d'une chaîne d'extension. Les caractères de la chaîne de caractères sont numérotés par ordre croissant en commençant par 0.

1CB3 adresse Exp String vers de

1CC3 ***************************** adresse Exp String vers de

1CC3 Token dans zone
1CC5 valable?
1CC7 retour si pas
1CC9 (Pointer Start Exp Buffer)
1CD0 augmenter hl
1CD1 de la longueur
1CD2 de la chaîne d'extension

1CDB ************************************** KM WAIT KEY

Les routines KM WAIT KEY et KM READ KEY travaillent de façon similaire à KM WAIT CHAR, mais seul le buffer clavier est interrogé.
La chaîne d'extension et le buffer Put Back ne sont pas pris en compte.

1CDB KM READ KEY
1CDE KM WAIT KEY

1CE1 *************************************** KM READ KEY

Voir KM WAIT KEY.

1CFB Caps Lock State
1D12 Shift Lock State
1D17 caps lock?
1D1A si pas sauter
1D1D toggle caps lock
1D27 KM GET CONTROLS
1D2B (Shift Lock State)
1D32 KM GET SHIFT
1D35 KM GET TRANSLATE

1D38 *************************************** KM GET STATE

examiner si touches CAPS-LOCK et SHIFT-LOCK ont été actionnées.

1D38 (Shift Lock State)

1D3C *** Set State

1D3C (Shift Lock State)

1D40 *********************** KM UPDATE KEY STATE MAP

1D40 Mulitihit contr. à B63F
1D43 touches enfoncées pendant examen
1D46 Scan Keyboard
1D4C isoler SHIFT/CTRL
1D4F Key 16...23
1D54 Mulitihit contr. à B63F
1D57 Key State Map
1D74 Test Break
1D86 Key State Map
1D8B (adresse de la table de répétition)

1D9E (KM Delay)

1DB8 ************************************ KM TEST BREAK

1DC1 KM BREAK EVENT
1DCE KM BREAK EVENT

1DE5 ************************************ KM GET JOYSTICK

L'état du joystick au moment du test est déterminé à l'aide de la Key State Map.

1DE5 (Joystick 1)
1DEB (Joystick 0)

1DF2 *************************************** KM GET DELAY

aller chercher paramètres pour emploi et vitesse de la répétition de touches.

1DF2 (KM Delay)

1DF6 *************************************** KM SET DELAY

fixer emploi et vitesse de répétition de touches.

1DF6 (Km Delay)

1DFA ************************************* KM ARM BREAK

autoriser la touche Break.

1DFA KM DISARM BREAK
1DFD Break Event Block
1E02 KL INIT EVENT

1E0B *********************************** KM DISARM BREAK

la touche Break est verrouillée.

1E13 KL DEL SYNCHRONOUS

1E19 ************************************ KM BREAK EVENT

exécuter routines lorsque la touche Break est actionnée.

1E24 KL EVENT

1E2F ************************************** KM GET REPEAT

Tester, pour une touche déterminée, s'il s'agit d'une touche avec fonction de répétition activée.

1E2F (adresse de la table de répétition)
1E32 fixer Z en fonction du bit touche

1E34 ** KM SET REPEAT

KM SET REPEAT vous permet de déterminer quelles touches doivent être dotées de la fonction de répétition.
Il faut placer en a le numéro de touche. b doit contenir &FF si la touche doit avoir une fonction de répétition et 0 s'il s'agit d'annuler la fonction de répétition de cette touche.

1E34 Key > 80?
1E36 oui alors incorrect
1E37 (adresse de la table de répétition)
1E3A aller chercher bit correspondant à la touche

1E45 ** KM TEST KEY

L'état de la Key State Map permet d'examiner si une touche ou un joystick a été activé.

1E46 (Key 16...23)
1E49 isoler SHIFT/CTRL
1E4D Key State Map
1E50 aller chercher bit correspondant à la touche
1E53 masquer bit touche

1E55 ****************************** aller chercher bit correspondant à la touche

1E57 Key#
1E59 /8
1E5F adresser Key Map
1E62 masques bits
1E65 charger
1E67 bit
1E68 correspondant
1E69 à la touche

1E6D ** masques bits

1E6D 01 02 04 08 10 20 40 80

1EC4 ********************************* KM GET TRANSLATE

aller chercher entrée du premier niveau de la table clavier (Key State Map).

1EC4 (adresse table traduction touche)
1EC7 Get Key Table

1EC9 ** KM GET SHIFT

aller chercher entrée du second niveau de la table clavier.

1EC9 (Adresse Key SHIFT Table)
1ECC Get Key Table

1ECE ************************************ KM GET CONTROL

aller chercher entrée du troisième niveau de la table clavier.

1ECE (Adresse Key CTRL Table)

1ED1 ** Get Key Table

1ED8 ********************************** KM SET TRANSLATE

effectuer une entrée dans le premier niveau de la table clavier.

1ED8 (adresse table traduction touche)
1EDB Set Key Table

1EDD ** KM SET SHIFT

effectuer une entrée dans le second niveau de la table clavier.

1EDD (Adresse Key SHIFT Table)
1EE0 Set Key Table

1EE2 ************************************** KM SET CONTROL

effectuer une entrée dans le troisième niveau de la table clavier.

1EE2 (Adresse Key CTRL Table)

1EE5 ** Set Key Table

1EEF ********************************** Key Translation Table

1EEF F0 F3 F1 89 86 83 8B 8A
1EF7 F2 E0 87 88 85 81 82 80
1EFF 10 5B 0D 5D 84 FF 5C FF
1F07 5E 2D 40 70 3B 3A 2F 2E
1F0F 30 39 6F 69 6C 6B 6D 2C
1F17 38 37 75 79 68 6A 6E 20
1F1F 36 35 72 74 67 66 62 76
1F27 34 33 65 77 73 64 63 78
1F2F 31 32 FC 71 09 61 FD 7A
1F37 0B 0A 08 09 58 5A FF 7F

1F3F ** Key SHIFT Table

1F3F F4 F7 F5 89 86 83 8B 8A

```
1F47   F6 E0 87 88 85 81 82 80
1F4F   10 7B 0D 7D 84 FF 60 FF
1F57   A3 3D 7C 50 2B 2A 3F 3E
1F5F   5F 29 4F 49 4C 4B 4D 3C
1F67   28 27 55 59 48 4A 4E 20
1F6F   26 25 52 54 47 46 42 56
1F77   24 23 45 57 53 44 43 58
1F7F   21 22 FC 51 09 41 FD 5A
1F87   0B 0A 08 09 58 5A FF 7F

1F8F ************************************** Key CTRL Table

1F8F   F8 FB F9 89 86 83 8C 8A
1F97   FA E0 87 88 85 81 82 80
1F9F   10 1B 0D 1D 84 FF 1C FF
1FA7   1E FF 00 10 FF FF FF FF
1FAF   1F FF 0F 09 0C 0B 0D FF
1FB7   FF FF 15 19 08 0A 0E FF
1FBF   FF FF 12 14 07 06 02 16
1FC7   FF FF 05 17 13 04 03 18
1FCF   FF 7E FC 11 E1 01 FE 1A
1FD7   FF FF FF FF FF FF FF 7F
1FDF   07 03 4B FF FF FF FF FF
1FE7   AB 8F
```

2.5.8 SOUND MANAGER (SOUND)

Il n'y a pas grand chose à dire sur ce pack, bien qu'il soit très puissant. La production du son proprement dite y prend en fait peu de place. La plus grande partie est occupée par la gestion des diverses files d'attente au rang desquelles figure également la réalisation de la TONE ENVELOPPE, que le PSG ne maîtrise pas de lui-même.

L'amateur de musique préfèrera sans doute programmer directement le PSG car les routines du SOUND sont trop taillées sur mesure pour les instructions Basic correspondantes. Pour jouer des mélodies, même à trois voix et même avec un tempo rapide, le Basic est très suffisant.

Mais si cela ne vous suffit pas, par exemple si vous voulez réaliser une bonne percussion (c'est-à-dire avec des changements de son importants), ce qui n'est qu'imparfaitement possible en Basic avec des sons brefs mais complexes. Il vous faut donc, dans ce cas, passer à la programmation en langage-machine.

1FE9 ** SOUND RESET

réinitialiser l'ensemble du SOUND MANAGER. Suppression de toutes les files d'attentes.

1FF3 Sound Event
1FF8 KL INIT EVENT
2000 paramètres SOUND canal A

2050 *** SOUND HOLD

arrêt de toutes les notes, peut être remis en cause par SOUND CONTINUE.

2050 act. activité SOUND
2058 canaux activés?
2059 si pas retour
205C volume
205E de tous les canaux
2060 sur 0
2063 MC SOUND REGISTER

206B ************************************** SOUND CONTINUE

traiter à nouveau les notes arrêtées auparavant (SOUND HOLD).

206B (ancienne activité
206E SOUND (après HOLD))
206F canal activé?
2070 si pas retour
2076 pour tous

2079 les canaux
207A fixer à nouveau
207D ancien volume

208B *** Sound Event

209D canal activé?
209F non alors suivant

20D7 ************************************** Scan Sound Queues

20D7 act. activité SOUND
2111 KL EVENT

2114 *** SOUND QUEUE

ajouter note à la file d'attente.

2114 SOUND CONTINUE

21AC ************************************** SOUND RELEASE

autoriser notes.

21AD SOUND CONTINUE

21CE ** SOUND CHECK

Y a-t-il encore de la place dans la file d'attente?

21EB *********************************** SOUND ARM EVENT

'armer' bloc event pour le cas où une place se libèrerait dans la file d'attente.

2206 KL EVENT
2258 act. activité SOUND
227D KL EVENT
2296 paramètres SOUND canal A

229E paramètres SOUND canal B
22A6 paramètres SOUND canal C
22B8 paramètres SOUND canal C
22C0 paramètres SOUND canal B
22F3 charger générateur de bruit
22F5 MC SOUND REGISTER
2303 courbes d'enveloppe de volume
2342 fixer volume
237D courbe d'enveloppe
237F MC SOUND REGISTER
2383 longueur courbe d'enveloppe Lo
2385 MC SOUND REGISTER
2389 longueur courbe d'enveloppe Hi
238B MC SOUND REGISTER
2390 fixer volume

23DB *** fixer volume

23E2 volume
23E4 MC SOUND REGISTER
23EF act. activité SOUND
2403 registre de commande de canal
2405 MC SOUND REGISTER
240C SOUND T ADRESS
2486 hauteur de note Lo
2489 MC SOUND REGISTER
248F hauteur de note Hi
2492 MC SOUND REGISTER

2495 ****************************** SOUND AMPL ENVELOPE

créer courbe d'enveloppe de volume (15 amplitudes différentes).

2495 courbes d'enveloppe de volume
2498 copier courbe d'enveloppe

249A ****************************** SOUND TONE ENVELOPE

créer courbe d'enveloppe de note (15 courbes d'enveloppe de note

différentes).

249A courbes d'enveloppe de note

249D************************************copiercourbed'enveloppe

249E aller chercher adresse courbe d'enveloppe

24A6 ************************************ SOUND A ADRESS

aller chercher adresse d'une courbe d'enveloppe.

24A6 courbes d'enveloppe de volume
24A9 aller chercher adresse courbe d'enveloppe

24AB ************************************ SOUND T ADRESS

aller chercher adresse d'une courbe d'enveloppe de note

24AB courbes d'enveloppe de note

24AE ******************************** aller chercher adresse courbe d'enveloppe

2.5.9 CASSETTE MANAGER (CAS)

Soyez sans crainte, nous n'avons pas oublié que votre ordinateur dispose d'un lecteur de disquette intégré, ce qui rend pour vous l'utilisation de cassettes pratiquement ou même tout à fait inutile. Nous ne vous en présentons pas moins le CASSETTE MANAGER qui dispose d'un certain nombre de routines qu'il est bon de connaître.

24BC ** CAS INITIALISE

initialisation complète du pack cassette

24BC CAS IN ABANDON
24C3 CAS NOISY

24CE ** CAS SET SPEED

fixer vitesse d'écriture.

24D9 (Cass. Speed)

24E1 *** CAS NOISY

messages cassette activés/désactivés. les messages d'erreur restent activés.

24E1 (Cass. Message Flag)

24E5 *** CAS IN OPEN

CAS IN OPEN ouvre un fichier d'entrée. Il faut pour cela placer en b la longueur du nom de fichier, en hl l'adresse de début du nom de fichier et en de l'adresse de début d'une zone de la Ram de 2 K qui sera utilisée comme buffer d'entrée.

Au retour de la routine, hl contient l'adresse de début de la téte de fichier (header).
a, bc et de contiennent d'autres valeurs tirées du header que vous pouvez cependant retirer vous-même directement du header, puisque vous disposez de l'adresse à laquelle il se trouve.

Les flags carry et zéro vous informent sur le succés de l'opération:
Carry=1 et zéro=0 signifient que tout a bien marché.
Carry=0 et zéro=0 signifient qu'il y a déjà un autre fichier d'ouvert.
Si la touche ESC a été enfoncée, carry=0 et zéro=1.

24E5 Input Buffer Status
24E9 Cass. Open
24ED lire File Header

24FE ** CAS OUT OPEN

CAS OUT OPEN ouvre un fichier en sortie. Les paramètres à transmettre et la signification des flags sont les mêmes que ci-dessus. Naturellement, de doit ici contenire l'adresse du buffer de sortie.

24FE Output Buffer Status

2502 ** Cass. Open

2550 ** CAS IN CLOSE

fermeture correcte du fichier d'entrée.

2550 (Input Buffer Status)

2557 ************************************** CAS IN ABANDON

interrompre immédiatement la lecture et fermer le fichier d'entrée (en cas d'erreur).

2557 Input Buffer Status

257F *************************************** CAS OUT CLOSE

fermeture correcte du fichier de sortie.

257F   (Output Buffer Status)

2599 ************************************ CAS OUT ABANDON

fermer immédiatement fichier de sortie et marquer le périphérique de sortie comme 'fermé'. les données non encore écrites sont détruites.

2599   Output Buffer Status

25A0 *** CAS IN CHAR

CAS IN CHAR va chercher un caractère dans le buffer d'entrée et le transmet à travers a. Si c'était le dernier caractère du buffer, un nouveau bloc est automatiquement lu sur la cassette.
Si carry=0 et zéro=0, c'est que la fin du fichier (EOF) a été atteinte ou que le fichier n'était pas ouvert. Les autres combinaisons ont le même sens que ci-dessus.

25A5   Check Input Buffer Status
25B0   lire File Header
25BC   (Pointer Input Buffer)
25BF   ld a,(hl)
25C1   (Pointer Input Buffer)

25C6 *** CAS OUT CHAR

CAS OUT CHAR écrit le caractère qui se trouve dans a dans le buffer de sortie. Si celui-ci est plein, le contenu du buffer est automatiquement écrit sur la cassette.
La signification des flags est la même que ci-dessus.

25CA   Output Buffer Status
25CF   Check Buffer Status
25EA   (Pointer Output Buffer)
25EF   (Pointer Output Buffer)

25F6 ******************************** Check Input Buffer Status

25F6 Input Buffer Status

25F9 ************************************** Check Buffer Status

2603 ** CAS TEST EOF

tester si fin de fichier atteinte.

2603 CAS IN CHAR

2607 *** CAS RETURN

renvoyer dernier caractère lu dans le buffer.

260F (Pointer Input Buffer)
2613 (Pointer Input Buffer)

2618 *** CAS IN DIRECT

transférer fichier d'entrée entier dans la mémoire, pas de lecture caractère par caractère.

261B (Check Input Buffer Status)
2631 lire File Header
263C (Adr. Start Input Buffer)
2647 KL LDIR CONT'D
2650 KL LDDR CONT'D

2653 ************************************* CAS OUT DIRECT

écrire zone mémoire définie sur cassette (pas à travers le buffer).

2656 Output Buffer Status
265B Check Buffer Status
266E (Adr. Start Output Buffer)
2685 (Adr. Start Output Buffer)

2692 ** CAS CATALOG

sortie du catalogue d'une cassette sur l'écran.

2692 Input Buffer Status
269C (Adr. Start Input Buffer)
26A1 CAS NOISY
26A9 CAS IN ABANDON

26AC ************************************ lire File Header

26C3 CAS READ
26E0 (Input Buffer Status)
26EF (Adr. Start Input Buffer)
26F2 (Pointer Input Buffer)
26F7 CAS READ
271B Input Buffer Status
2743 (File Header Input)
274E File Header Input
2760 File Header Input
277B CAS OUT CLOSE
2781 CAS MOTOR STOP
2790 File Header Output
279E (Adr. Start Output Buffer)
27A1 (Pointer Output Buffer)
27A8 File Header Output
27B0 CAS WRITE
27BC CAS WRITE
27D9 Output Buffer Status
27F5 CAS START MOTOR
2807 (Cass. Message Flag)
2846 TXT WR CHAR
2871 sortir message CAS (1 caractère)
2886 sortir message CAS (# in b)
288C sortir message CAS (1 caractère)

2891 *********************** sortir message CAS (# in b)

2891 TXT GET CURSOR
289D messages cassette
28C9 sortir message CAS (1 caractère)

28D0 sortir message CAS (1 caractère)
28D2 (Cass. Message Flag)
28D8 sortir message CAS (# in b)
28DB KM READ CHAR
28DE TXT CUR ON
28E1 KM WAIT KEY
28E4 TXT CUR OFF

28F0 ********************* sortir message CAS (1 caractère)

28F0 TXT OUTPUT
28F7 TXT SET COLUMN
28FE TXT GET WINDOW
2902 TXT GET CURSOR
2924 sortir message CAS (1 caractère)
292F (Input Buffer Status)

2935 ************************************ messages cassette

2935 Press
293B PLAY
293F then
2943 any
2946 key
294B error
2955 REC
2958 and
295D Read
2963 Write
296A Rewind
2970 tape
2975 Found
297D Loading
2985 Saving
298D ok
2990 block
2996 Unnamed
299D file

29A6 *** CAS READ

lire un bloc de la cassette. cette routine est appelée par des routines de plus haut rang.

29A6 moteur activé & ouvrir clavier

29AF ** CAS WRITE

écrire un bloc sur cassette. est appelé, comme CAS READ, par des routines de plus haut rang.

29AF moteur activé & ouvrir clavier

29C1 *** CAS CHECK

comparer bloc sur la bande avec contenu de la mémoire.

29C1 moteur activé & ouvrir clavier
29D2 Port A=Out
29D7 moteur activé
29DE CAS RESTORE MOTOR

29E3 ****************************** moteur activé & ouvrir clavier

29EA SOUND RESET
29F0 CAS START MOTOR
29F4 Sound I/O Port select
29F9 Strobe activé
29FE Strobe désactivé
2A02 Port A=In
2A07 ouvrir clavier Y9 (ESC)
2A0A & Sound I/O sur port A
2A3C RAM LAM (IX)
2A67 RAM LAM (IX)
2A95 Cass. Input RD DATA & Test ESC
2A9D Cass. Input RD DATA & Test ESC
2AB2 Cass. Input RD DATA & Test ESC

2B3D ********************** Cass. Input RD DATA & Test ESC

2B3D Port A
2B3F Keyb. X
2B41 ESC?
2B43 si oui retour
2B4D Port B
2B55 Input RD DATA
2B8E WR DATA désactivé
2B90 Cass. Output WR DATA
2B9F WR DATA activé
2BA1 Cass. Output WR DATA

2BA7 ********************************** Cass. Output WR DATA

2BB1 Port Control
2BB3 WR DATA

2BBB ************************************ CAS START MOTOR

moteur cassette activé

2BBD CAS RESTORE MOTOR

2BBF ************************************** CAS STOP MOTOR

arrêter moteur cassette.

2BC1 ********************************** CAS RESTORE MOTOR

rétablit l'ancien état du moteur. après mise en marche du moteur on attend que le nombre de rotations requis soit atteint.

2BC2 Port C
2BCE moteur activé/désactivé
2BE9 KM TEST KEY

2.5.10 SCREEN EDITOR (EDIT)

L'éditeur n'est pas en réalité un pack dans le sens où nous l'avons compris jusqu'ici. Il n'est en effet pas du tout utilisé par le système d'exploitation.
Il doit plutôt être considéré comme lié aux packs arithmétiques. De même que ceux-ci, l'éditeur n'est appelé que par le Basic.

Nous ne voyons pas quelles routines individuelles pourraient être utilisées, si ce n'est tout au plus l'éditeur lui-même globalement.
Il vous faut pour cela fournir à hl l'adresse de début du texte que vous souhaitez éditer. Ce texte doit comprendre un maximum de 255 caractères, ce qui correspond également à la taille maximum d'une ligne Basic.

2C02 ** EDIT

2C12 EDIT exécuter saut
2C1A EDIT exécuter saut
2C1D pointeur sur buffer d'entrée
2C1E compter caractères dans buffer
2C24 (Insert Flag)
2C2D caractère de clavier

2C42 *********************************** EDIT exécuter saut

2C49 EDIT table de saut 1
2C4E caractère dans le buffer?
2C50 sauter si oui
2C52 une des touches curseur?
2C54 sauter si pas
2C56 touche curseur et SHFT/CTRL?
2C58 sauter si oui
2C5A EDIT table de saut 2

2C72 *********************************** EDIT table de saut 1

```
2C72   db 13   nombre entrées

2C73   dw 2D8A   ajouter caractère

2C75   db FC
2C76   dw 2CD0   ESC

2C78   db EF
2C79   dw 2CCE   aucun effet

2C7B   db 0D
2C7C   dw 2CF2   ENTER

2C7E   db F0
2C7F   dw 2D3C   CRSR UP (buffer)

2C81   db F1
2C82   dw 2D0A   CRSR DWN (buffer)

2C84   db F2
2C85   dw 2D34   CRSR LEFT (buffer)

2C87   db F3
2C88   dw 2D02   CRSR RGHT (buffer)

2C8A   db F8
2C8B   dw 2D4F   CTRL & CRSR UP

2C8D   db F9
2C8E   dw 2D1D   CTRL & CRSR DWN

2C90   db FA
2C91   dw 2D45   CTRL & CRSR LEFT

2C93   db FB
2C94   dw 2D14   CTRL & CRSR RGHT

2C96   db F4
2C97   dw 2E21   SHFT & CRSR UP
```

2C99   db F5
2C9A   dw 2E26   SHFT & CRSR DWN

2C9C   db F6
2C9D   dw 2E1C   SHFT & CRSR LEFT

2C9F   db F7
2CA0   dw 2E17   SHFT & CRSR RGHT

2CA2   db E0
2CA3   dw 2E65   COPY

2CA5   db 7F
2CA6   dw 2DC3   DEL

2CA8   db 10
2CA9   dw 2DCD   CLR

2CAB   db E1
2CAC   dw 2D81   CTRL & TAB (Flip Insert)

2CAE ************************************ EDIT table de saut 2

2CAE   db 04   nombre entrées

2CAF   dw 2CFE   bip-bip

2CB1   db F0
2CB2   dw 2CBD   CRSR UP

2CB4   db F1
2CB5   dw 2CC1   CRSR DWN

2CB7   db F2
2CB8   dw 2CC9   CRSR LEFT

2CBA   db F3
2CBB   dw 2CC5   CRSR RGHT

2CBD ** CRSR UP

2CC1 ** CRSR DWN

2CC5 ** CRSR RGHT

2CC9 ** CRSR LEFT

2CCB TXT OUTPUT

2CD0 ** ESC

2CD0 ENTER
2CD4 message *BREAK*
2CD7 ENTER
2CDA TXT GET CURSOR
2CE0 CR
2CE2 TXT OUTPUT
2CE5 CRSR DWN

2CEA *********************************** message *BREAK*

2CEA 2A 42 72 65 61 6B 2A 00 *BREAK*

2CF1 ** ENTER

2CFC mettre le flag de retenue

2CFE ** BIP-BIP

2CFE BEL

2D02 *********************************** CRSR RGHT (buffer)

2D07 BIP-BIP

2D0A *********************************** CRSR DWN (buffer)
2D10 BIP-BIP

2D14 ********************************* CTRL & CRSR RGHT

2D1D ********************************* CTRL & CRSR DWN

2D34 ********************************** CRSR LEFT (buffer)

2D39 BIP-BIP

2D3C ************************************* CRSR UP (buffer)

2D41 BIP-BIP

2D45 ********************************* CTRL & CRSR LEFT

2D4F ************************************* CTRL & CRSR UP

2D74 TXT GET WINDOW
2D7B TXT GET CURSOR

2D81 ****************************** CTRL & TAB (Flip insert)

2D81 (Insert Flag)
2D85 (Insert Flag)

2D8A *************************************** ajouter caractère

2D8D (Insert Flag)
2DA1 BIP-BIP

2DC3 ** DEL

2DC8 BIP-BIP

2DCD ** CLR

2DCF BIP-BIP
2E0E TXT VALIDATE

2E17 ********************************* SHFT & CRSR RGHT

2E1C ********************************** SHFT & CRSR LEFT

2E21 ************************************* SHFT & CRSR UP

2E26 *********************************** SHFT & CRSR DWN

2E2E TXT GET CURSOR
2E37 TXT VALIDATE
2E4A TXT PLACE/REMOVE CURSOR
2E4F TXT PLACE/REMOVE CURSOR
2E57 TXT GET CURSOR
2E5B TXT SET CURSOR
2E62 TXT SET CURSOR

2E65 *** COPY

2E67 TXT GET CURSOR
2E74 TXT GET CURSOR
2E7C TXT SET CURSOR
2E7F TXT PLACE/REMOVE CURSOR
2E82 TXT RD CHAR
2E87 TXT SET CURSOR
2E8E TXT VALIDATE
2E9C ajouter caractère
2E9F BIP-BIP
2ED3 TXT GET CURSOR
2ED9 TXT VALIDATE
2EDD TXT OUTPUT
2EE7 TXT GET CURSOR
2EF4 TXT GET CURSOR
2EFB TXT SET CURSOR
2F07 TXT GET CURSOR
2F0E TXT VALIDATE
2F19 TXT VALIDATE

2F2A   TXT GET CURSOR
2F2F   TXT VALIDATE
2F3C   TXT WR CHAR
2F40   TXT GET CURSOR

2F56 *********************************** caractère de clavier

2F56   TXT GET CURSOR
2F5A   TXT VALIDATE
2F60   KM WAIT CHAR
2F63   TXT CUR ON
2F66   TXT GET CURSOR
2F6D   KM WAIT CHAR
2F70   TXT CUR OFF

*****   BD97 PI
2F73
2F78   PI
*****   BD5E copier variable
2F91
*****   BD64 valeur 4 octets en virgule flottante
2FC8
*****   BDB5 valeur 4 octets fois 256 en entier
2FD1
*****   BD67 virgule flottante en entier
2FD9
*****   BD6A virgule flottante en entier
3001
*****   BD6D FIX
3014
*****   BD70 INT
3055
*****   BD73
305F
*****   BD76 multiplier nombre par 10^A
30C6
*****   BDB8 RND INIT
3136
*****   BDBB SET RANDOM SEED

3143
***** BD7C RND
3159
***** BD88 aller chercher dernière valeur RND
3188
***** BDA3 LOG10
31B1
***** BDA0 LOG
31B6
***** BDA6 EXP
322F
***** BD9A EXP
32AC
***** BD9D élévation à la puissance
32AF
***** BD94 DEG/RAD
3345
***** BDAA COS
3349
***** BDA7 SIN
3353
***** BDAF TAN
33C8
***** BDB2 ATN
33D8
***** BD7F soustraction
349E
***** BD79 addition
34A2
***** BD82 multiplication
3577
***** BD85 division
3604
***** BD8B comparaison
36DF
***** BD91 SGN
3727
***** BD8E changement de signe
3731

2.6 Le générateur de caractères

Ce n'est pas que nous voulions à tout prix abuser de votre patience avec les pages suivantes ni que nous pensions que l'ouvrage ne comporte pas encore assez de pages.
Nous pensons simplement que le jeu de caractères est un outil de travail important auquel s'appliquent même spécialement certaines instructions du jeu d'instructions Basic.

Pour que vous n'ayez pas à réinventer la poudre chaque fois que vous utilisez ces instructions, par exemple lorsque vous voulez produire des accents, il vous suffit de rechercher la forme du 'e' et de rajouter au dessus les points qui formeront l'accent aigu ou grave. Il vous suffit alors d'utiliser les valeurs ainsi calculées dans votre instruction de définition d'un caractère.

Nous nous permettons de vous donner un petit conseil. Vous constaterez que la plupart des dessins figurant dans les pages suivantes marquent toujours les lignes verticales par une paire de points (deux points sur la même ligne horizontale). Il vaut mieux éviter en effet de constituer des lignes verticales n'ayant qu'un point de largeur. En effet un point isolé est difficile à discerner à l'écran, surtout si vous disposez d'un moniteur couleur.

Mais maintenant vous pouvez donner libre cours à votre imagination et faire vos propres expériences. N'oubliez pas notre conseil de toujours former les lignes verticales avec des paires de pixels. Avant de vous lancer dans une redéfinition de caractères, vous pouvez toutefois chercher si, parmi les 256 caractères du CHARACTER GENERATOR du CPC664/6128, vous n'en trouvez pas un qui vous convienne.

CHARACTERS

Address	Data
3800	FF
3801	C3
3802	C3
3803	C3
3804	C3
3805	C3
3806	C3
3807	FF
3808	FF
3809	C0
380A	C0
380B	C0
380C	C0
380D	C0
380E	C0
380F	C0
3810	18
3811	18
3812	18
3813	18
3814	18
3815	18
3816	18
3817	FF
3818	03
3819	03
381A	03
381B	03
381C	03
381D	03
381E	03
381F	FF
3820	0C
3821	18
3822	30
3823	7E
3824	0C
3825	18
3826	30
3827	00
3828	FF
3829	C3
382A	E7
382B	DB
382C	DB
382D	E7
382E	C3
382F	FF
3830	00
3831	01
3832	03
3833	06
3834	CC
3835	78
3836	30
3837	00
3838	3C
3839	66
383A	C3
383B	C3
383C	FF
383D	24
383E	E7
383F	00
3840	00
3841	00
3842	30
3843	60
3844	FF
3845	60
3846	30
3847	00
3848	00
3849	00
384A	0C
384B	06
384C	FF
384D	06
384E	0C
384F	00
3850	18
3851	18
3852	18
3853	18
3854	DB
3855	7E
3856	3C
3857	18
3858	18
3859	3C
385A	7E
385B	DB
385C	18
385D	18
385E	18
385F	18

CHARACTERS

Address	Data	Address	Data
3860	18	3868	00
3861	5A	3869	03
3862	3C	386A	33
3863	99	386B	63
3864	DB	386C	FE
3865	7E	386D	60
3866	3C	386E	30
3867	18	386F	00
3870	3C	3878	3C
3871	66	3879	66
3872	FF	387A	C3
3873	DB	387B	DB
3874	DB	387C	DB
3875	FF	387D	C3
3876	66	387E	66
3877	3C	387F	3C
3880	FF	3888	3C
3881	C3	3889	7E
3882	C3	388A	DB
3883	FF	388B	DB
3884	C3	388C	DF
3885	C3	388D	C3
3886	C3	388E	66
3887	FF	388F	3C
3890	3C	3898	3C
3891	66	3899	66
3892	C3	389A	C3
3893	DF	389B	FB
3894	DB	389C	DB
3895	DB	389D	DB
3896	7E	389E	7E
3897	3C	389F	3C
38A0	3C	38A8	00
38A1	7E	38A9	01
38A2	DB	38AA	33
38A3	DB	38AB	1E
38A4	FB	38AC	CE
38A5	C3	38AD	7B
38A6	66	38AE	31
38A7	3C	38AF	00
38B0	7E	38B8	03
38B1	66	38B9	03
38B2	66	38BA	03
38B3	66	38BB	FF
38B4	66	38BC	03
38B5	66	38BD	03
38B6	66	38BE	03
38B7	E7	38BF	00

CHARACTERS

Address	Value
38C0	FF
38C1	66
38C2	3C
38C3	18
38C4	18
38C5	3C
38C6	66
38C7	FF
38C8	18
38C9	18
38CA	3C
38CB	3C
38CC	3C
38CD	3C
38CE	18
38CF	18
38D0	3C
38D1	66
38D2	66
38D3	30
38D4	18
38D5	00
38D6	18
38D7	00
38D8	3C
38D9	66
38DA	C3
38DB	FF
38DC	C3
38DD	C3
38DE	66
38DF	3C
38E0	FF
38E1	DB
38E2	DB
38E3	DB
38E4	FB
38E5	C3
38E6	C3
38E7	FF
38E8	FF
38E9	C3
38EA	C3
38EB	FB
38EC	DB
38ED	DB
38EE	DB
38EF	FF
38F0	FF
38F1	C3
38F2	C3
38F3	DF
38F4	DB
38F5	DB
38F6	DB
38F7	FF
38F8	FF
38F9	DB
38FA	DB
38FB	DB
38FC	DF
38FD	C3
38FE	C3
38FF	FF
3900	00
3901	00
3902	00
3903	00
3904	00
3905	00
3906	00
3907	00
3908	18
3909	18
390A	18
390B	18
390C	18
390D	00
390E	18
390F	00
3910	6C
3911	6C
3912	6C
3913	00
3914	00
3915	00
3916	00
3917	00
3918	6C
3919	6C
391A	FE
391B	6C
391C	FE
391D	6C
391E	6C
391F	00

CHARACTERS

Address	Data
3920	18
3921	3E
3922	58
3923	3C
3924	1A
3925	7C
3926	18
3927	00
3928	00
3929	C6
392A	CC
392B	18
392C	30
392D	66
392E	C6
392F	00
3930	38
3931	6C
3932	38
3933	76
3934	DC
3935	CC
3936	76
3937	00
3938	18
3939	18
393A	30
393B	00
393C	00
393D	00
393E	00
393F	00
3940	0C
3941	18
3942	30
3943	30
3944	30
3945	18
3946	0C
3947	00
3948	30
3949	18
394A	0C
394B	0C
394C	0C
394D	18
394E	30
394F	00
3950	00
3951	66
3952	3C
3953	FF
3954	3C
3955	66
3956	00
3957	00
3958	00
3959	18
395A	18
395B	7E
395C	18
395D	18
395E	00
395F	00
3960	00
3961	00
3962	00
3963	00
3964	00
3965	18
3966	18
3967	30
3968	00
3969	00
396A	00
396B	7E
396C	00
396D	00
396E	00
396F	00
3970	00
3971	00
3972	00
3973	00
3974	00
3975	18
3976	18
3977	00
3978	06
3979	0C
397A	18
397B	30
397C	60
397D	C0
397E	80
397F	00

CHARACTERS

Address	Value
3980	7C
3981	C6
3982	CE
3983	D6
3984	E6
3985	C6
3986	7C
3987	00
3988	18
3989	38
398A	18
398B	18
398C	18
398D	18
398E	7E
398F	00
3990	3C
3991	66
3992	06
3993	3C
3994	60
3995	66
3996	7E
3997	00
3998	3C
3999	66
399A	06
399B	1C
399C	06
399D	66
399E	3C
399F	00
39A0	1C
39A1	3C
39A2	6C
39A3	CC
39A4	FE
39A5	0C
39A6	1E
39A7	00
39A8	7E
39A9	62
39AA	60
39AB	7C
39AC	06
39AD	66
39AE	3C
39AF	00
39B0	3C
39B1	66
39B2	60
39B3	7C
39B4	66
39B5	66
39B6	3C
39B7	00
39B8	7E
39B9	66
39BA	06
39BB	0C
39BC	18
39BD	18
39BE	18
39BF	00
39C0	3C
39C1	66
39C2	66
39C3	3C
39C4	66
39C5	66
39C6	3C
39C7	00
39C8	3C
39C9	66
39CA	66
39CB	3E
39CC	06
39CD	66
39CE	3C
39CF	00
39D0	00
39D1	00
39D2	18
39D3	18
39D4	00
39D5	18
39D6	18
39D7	00
39D8	00
39D9	00
39DA	18
39DB	18
39DC	00
39DD	18
39DE	18
39DF	30

Address	Value
39E0	0C
39E1	18
39E2	30
39E3	60
39E4	30
39E5	18
39E6	0C
39E7	00

Address	Value
39E8	00
39E9	00
39EA	7E
39EB	00
39EC	00
39ED	7E
39EE	00
39EF	00

Address	Value
39F0	60
39F1	30
39F2	18
39F3	0C
39F4	18
39F5	30
39F6	60
39F7	00

Address	Value
39F8	3C
39F9	66
39FA	66
39FB	0C
39FC	18
39FD	00
39FE	18
39FF	00

Address	Value
3A00	7C
3A01	C6
3A02	DE
3A03	DE
3A04	DE
3A05	C0
3A06	7C
3A07	00

Address	Value
3A08	18
3A09	3C
3A0A	66
3A0B	66
3A0C	7E
3A0D	66
3A0E	66
3A0F	00

Address	Value
3A10	FC
3A11	66
3A12	66
3A13	7C
3A14	66
3A15	66
3A16	FC
3A17	00

Address	Value
3A18	3C
3A19	66
3A1A	C0
3A1B	C0
3A1C	C0
3A1D	66
3A1E	3C
3A1F	00

Address	Value
3A20	F8
3A21	6C
3A22	66
3A23	66
3A24	66
3A25	6C
3A26	F8
3A27	00

Address	Value
3A28	FE
3A29	62
3A2A	68
3A2B	78
3A2C	68
3A2D	62
3A2E	FE
3A2F	00

Address	Value
3A30	FE
3A31	62
3A32	68
3A33	78
3A34	68
3A35	60
3A36	F0
3A37	00

Address	Value
3A38	3C
3A39	66
3A3A	C0
3A3B	C0
3A3C	CE
3A3D	66
3A3E	3E
3A3F	00

Address	Value
3A40	66
3A41	66
3A42	66
3A43	7E
3A44	66
3A45	66
3A46	66
3A47	00
3A48	7E
3A49	18
3A4A	18
3A4B	18
3A4C	18
3A4D	18
3A4E	7E
3A4F	00
3A50	1E
3A51	0C
3A52	0C
3A53	0C
3A54	CC
3A55	CC
3A56	78
3A57	00
3A58	E6
3A59	66
3A5A	6C
3A5B	78
3A5C	6C
3A5D	66
3A5E	E6
3A5F	00
3A60	F0
3A61	60
3A62	60
3A63	60
3A64	62
3A65	66
3A66	FE
3A67	00
3A68	C6
3A69	EE
3A6A	FE
3A6B	FE
3A6C	D6
3A6D	C6
3A6E	C6
3A6F	00
3A70	C6
3A71	E6
3A72	F6
3A73	DE
3A74	CE
3A75	C6
3A76	C6
3A77	00
3A78	38
3A79	6C
3A7A	C6
3A7B	C6
3A7C	C6
3A7D	6C
3A7E	38
3A7F	00
3A80	FC
3A81	66
3A82	66
3A83	7C
3A84	60
3A85	60
3A86	F0
3A87	00
3A88	38
3A89	6C
3A8A	C6
3A8B	C6
3A8C	DA
3A8D	CC
3A8E	76
3A8F	00
3A90	FC
3A91	66
3A92	66
3A93	7C
3A94	6C
3A95	66
3A96	E6
3A97	00
3A98	3C
3A99	66
3A9A	60
3A9B	3C
3A9C	06
3A9D	66
3A9E	3C
3A9F	00

Address	Value	Address	Value
3AA0	7E	3AA8	66
3AA1	5A	3AA9	66
3AA2	18	3AAA	66
3AA3	18	3AAB	66
3AA4	18	3AAC	66
3AA5	18	3AAD	66
3AA6	3C	3AAE	3C
3AA7	00	3AAF	00
3AB0	66	3AB8	C6
3AB1	66	3AB9	C6
3AB2	66	3ABA	C6
3AB3	66	3ABB	D6
3AB4	66	3ABC	FE
3AB5	3C	3ABD	EE
3AB6	18	3ABE	C6
3AB7	00	3ABF	00
3AC0	C6	3AC8	66
3AC1	6C	3AC9	66
3AC2	38	3ACA	66
3AC3	38	3ACB	3C
3AC4	6C	3ACC	18
3AC5	C6	3ACD	18
3AC6	C6	3ACE	3C
3AC7	00	3ACF	00
3AD0	FE	3AD8	3C
3AD1	C6	3AD9	30
3AD2	8C	3ADA	30
3AD3	18	3ADB	30
3AD4	32	3ADC	30
3AD5	66	3ADD	30
3AD6	FE	3ADE	3C
3AD7	00	3ADF	00
3AE0	C0	3AE8	3C
3AE1	60	3AE9	0C
3AE2	30	3AEA	0C
3AE3	18	3AEB	0C
3AE4	0C	3AEC	0C
3AE5	06	3AED	0C
3AE6	02	3AEE	3C
3AE7	00	3AEF	00
3AF0	18	3AF8	00
3AF1	3C	3AF9	00
3AF2	7E	3AFA	00
3AF3	18	3AFB	00
3AF4	18	3AFC	00
3AF5	18	3AFD	00
3AF6	18	3AFE	00
3AF7	00	3AFF	FF

CHARACTERS

Address	Data
3B00	30
3B01	18
3B02	0C
3B03	00
3B04	00
3B05	00
3B06	00
3B07	00
3B08	00
3B09	00
3B0A	78
3B0B	0C
3B0C	7C
3B0D	CC
3B0E	76
3B0F	00
3B10	E0
3B11	60
3B12	7C
3B13	66
3B14	66
3B15	66
3B16	DC
3B17	00
3B18	00
3B19	00
3B1A	3C
3B1B	66
3B1C	60
3B1D	66
3B1E	3C
3B1F	00
3B20	1C
3B21	0C
3B22	7C
3B23	CC
3B24	CC
3B25	CC
3B26	76
3B27	00
3B28	00
3B29	00
3B2A	3C
3B2B	66
3B2C	7E
3B2D	60
3B2E	3C
3B2F	00
3B30	1C
3B31	36
3B32	30
3B33	78
3B34	30
3B35	30
3B36	78
3B37	00
3B38	00
3B39	00
3B3A	3E
3B3B	66
3B3C	66
3B3D	3E
3B3E	06
3B3F	7C
3B40	E0
3B41	60
3B42	6C
3B43	76
3B44	66
3B45	66
3B46	E6
3B47	00
3B48	18
3B49	00
3B4A	38
3B4B	18
3B4C	18
3B4D	18
3B4E	3C
3B4F	00
3B50	06
3B51	00
3B52	0E
3B53	06
3B54	06
3B55	66
3B56	66
3B57	3C
3B58	E0
3B59	60
3B5A	66
3B5B	6C
3B5C	78
3B5D	6C
3B5E	E6
3B5F	00

CHARACTERS

Address	Data
3B60	38
3B61	18
3B62	18
3B63	18
3B64	18
3B65	18
3B66	3C
3B67	00
3B68	00
3B69	00
3B6A	6C
3B6B	FE
3B6C	D6
3B6D	D6
3B6E	C6
3B6F	00
3B70	00
3B71	00
3B72	DC
3B73	66
3B74	66
3B75	66
3B76	66
3B77	00
3B78	00
3B79	00
3B7A	3C
3B7B	66
3B7C	66
3B7D	66
3B7E	3C
3B7F	00
3B80	00
3B81	00
3B82	DC
3B83	66
3B84	66
3B85	7C
3B86	60
3B87	F0
3B88	00
3B89	00
3B8A	76
3B8B	CC
3B8C	CC
3B8D	7C
3B8E	0C
3B8F	1E
3B90	00
3B91	00
3B92	DC
3B93	76
3B94	60
3B95	60
3B96	F0
3B97	00
3B98	00
3B99	00
3B9A	3C
3B9B	60
3B9C	3C
3B9D	06
3B9E	7C
3B9F	00
3BA0	30
3BA1	30
3BA2	7C
3BA3	30
3BA4	30
3BA5	36
3BA6	1C
3BA7	00
3BA8	00
3BA9	00
3BAA	66
3BAB	66
3BAC	66
3BAD	66
3BAE	3E
3BAF	00
3BB0	00
3BB1	00
3BB2	66
3BB3	66
3BB4	66
3BB5	3C
3BB6	18
3BB7	00
3BB8	00
3BB9	00
3BBA	C6
3BBB	D6
3BBC	D6
3BBD	FE
3BBE	6C
3BBF	00

CHARACTERS

Address	Value
3BC0	00
3BC1	00
3BC2	C6
3BC3	6C
3BC4	38
3BC5	6C
3BC6	C6
3BC7	00
3BD0	00
3BD1	00
3BD2	7E
3BD3	4C
3BD4	18
3BD5	32
3BD6	7E
3BD7	00
3BE0	18
3BE1	18
3BE2	18
3BE3	18
3BE4	18
3BE5	18
3BE6	18
3BE7	00
3BF0	76
3BF1	DC
3BF2	00
3BF3	00
3BF4	00
3BF5	00
3BF6	00
3BF7	00
3C00	00
3C01	00
3C02	00
3C03	00
3C04	00
3C05	00
3C06	00
3C07	00
3C10	0F
3C11	0F
3C12	0F
3C13	0F
3C14	00
3C15	00
3C16	00
3C17	00

Address	Value
3BC8	00
3BC9	00
3BCA	66
3BCB	66
3BCC	66
3BCD	3E
3BCE	06
3BCF	7C
3BD8	0E
3BD9	18
3BDA	18
3BDB	70
3BDC	18
3BDD	18
3BDE	0E
3BDF	00
3BE8	70
3BE9	18
3BEA	18
3BEB	0E
3BEC	18
3BED	18
3BEE	70
3BEF	00
3BF8	CC
3BF9	33
3BFA	CC
3BFB	33
3BFC	CC
3BFD	33
3BFE	CC
3BFF	33
3C08	F0
3C09	F0
3C0A	F0
3C0B	F0
3C0C	00
3C0D	00
3C0E	00
3C0F	00
3C18	FF
3C19	FF
3C1A	FF
3C1B	FF
3C1C	00
3C1D	00
3C1E	00
3C1F	00

CHARACTERS

Address	Data	Address	Data
3C20	00	3C28	F0
3C21	00	3C29	F0
3C22	00	3C2A	F0
3C23	00	3C2B	F0
3C24	F0	3C2C	F0
3C25	F0	3C2D	F0
3C26	F0	3C2E	F0
3C27	F0	3C2F	F0
3C30	0F	3C38	FF
3C31	0F	3C39	FF
3C32	0F	3C3A	FF
3C33	0F	3C3B	FF
3C34	F0	3C3C	F0
3C35	F0	3C3D	F0
3C36	F0	3C3E	F0
3C37	F0	3C3F	F0
3C40	00	3C48	F0
3C41	00	3C49	F0
3C42	00	3C4A	F0
3C43	00	3C4B	F0
3C44	0F	3C4C	0F
3C45	0F	3C4D	0F
3C46	0F	3C4E	0F
3C47	0F	3C4F	0F
3C50	0F	3C58	FF
3C51	0F	3C59	FF
3C52	0F	3C5A	FF
3C53	0F	3C5B	FF
3C54	0F	3C5C	0F
3C55	0F	3C5D	0F
3C56	0F	3C5E	0F
3C57	0F	3C5F	0F
3C60	00	3C68	F0
3C61	00	3C69	F0
3C62	00	3C6A	F0
3C63	00	3C6B	F0
3C64	FF	3C6C	FF
3C65	FF	3C6D	FF
3C66	FF	3C6E	FF
3C67	FF	3C6F	FF
3C70	0F	3C78	FF
3C71	0F	3C79	FF
3C72	0F	3C7A	FF
3C73	0F	3C7B	FF
3C74	FF	3C7C	FF
3C75	FF	3C7D	FF
3C76	FF	3C7E	FF
3C77	FF	3C7F	FF

CHARACTERS

Address	Value
3C80	00
3C81	00
3C82	00
3C83	18
3C84	18
3C85	00
3C86	00
3C87	00

Address	Value
3C88	18
3C89	18
3C8A	18
3C8B	18
3C8C	18
3C8D	00
3C8E	00
3C8F	00

Address	Value
3C90	00
3C91	00
3C92	00
3C93	1F
3C94	1F
3C95	00
3C96	00
3C97	00

Address	Value
3C98	18
3C99	18
3C9A	18
3C9B	1F
3C9C	0F
3C9D	00
3C9E	00
3C9F	00

Address	Value
3CA0	00
3CA1	00
3CA2	00
3CA3	18
3CA4	18
3CA5	18
3CA6	18
3CA7	18

Address	Value
3CA8	18
3CA9	18
3CAA	18
3CAB	18
3CAC	18
3CAD	18
3CAE	18
3CAF	18

Address	Value
3CB0	00
3CB1	00
3CB2	00
3CB3	0F
3CB4	1F
3CB5	18
3CB6	18
3CB7	18

Address	Value
3CB8	18
3CB9	18
3CBA	18
3CBB	1F
3CBC	1F
3CBD	18
3CBE	18
3CBF	18

Address	Value
3CC0	00
3CC1	00
3CC2	00
3CC3	F8
3CC4	F8
3CC5	00
3CC6	00
3CC7	00

Address	Value
3CC8	18
3CC9	18
3CCA	18
3CCB	F8
3CCC	F0
3CCD	00
3CCE	00
3CCF	00

Address	Value
3CD0	00
3CD1	00
3CD2	00
3CD3	FF
3CD4	FF
3CD5	00
3CD6	00
3CD7	00

Address	Value
3CD8	18
3CD9	18
3CDA	18
3CDB	FF
3CDC	FF
3CDD	00
3CDE	00
3CDF	00

Address	Data
3CE0	00
3CE1	00
3CE2	00
3CE3	F0
3CE4	F8
3CE5	18
3CE6	18
3CE7	18
3CE8	18
3CE9	18
3CEA	18
3CEB	F8
3CEC	F8
3CED	18
3CEE	18
3CEF	18
3CF0	00
3CF1	00
3CF2	00
3CF3	FF
3CF4	FF
3CF5	18
3CF6	18
3CF7	18
3CF8	18
3CF9	18
3CFA	18
3CFB	FF
3CFC	FF
3CFD	18
3CFE	18
3CFF	18
3D00	10
3D01	38
3D02	6C
3D03	C6
3D04	00
3D05	00
3D06	00
3D07	00
3D08	0C
3D09	18
3D0A	30
3D0B	00
3D0C	00
3D0D	00
3D0E	00
3D0F	00
3D10	66
3D11	66
3D12	00
3D13	00
3D14	00
3D15	00
3D16	00
3D17	00
3D18	3C
3D19	66
3D1A	60
3D1B	F8
3D1C	60
3D1D	66
3D1E	FE
3D1F	00
3D20	38
3D21	44
3D22	BA
3D23	A2
3D24	BA
3D25	44
3D26	38
3D27	00
3D28	7E
3D29	F4
3D2A	F4
3D2B	74
3D2C	34
3D2D	34
3D2E	34
3D2F	00
3D30	1E
3D31	30
3D32	38
3D33	6C
3D34	38
3D35	18
3D36	F0
3D37	00
3D38	18
3D39	18
3D3A	0C
3D3B	00
3D3C	00
3D3D	00
3D3E	00
3D3F	00

CHARACTERS

Address	Value	Address	Value
3D40	40	3D48	40
3D41	C0	3D49	C0
3D42	44	3D4A	4C
3D43	4C	3D4B	52
3D44	54	3D4C	44
3D45	1E	3D4D	08
3D46	04	3D4E	1E
3D47	00	3D4F	00
3D50	E0	3D58	00
3D51	10	3D59	18
3D52	62	3D5A	18
3D53	16	3D5B	7E
3D54	EA	3D5C	18
3D55	0F	3D5D	18
3D56	02	3D5E	7E
3D57	00	3D5F	00
3D60	18	3D68	00
3D61	18	3D69	00
3D62	00	3D6A	00
3D63	7E	3D6B	7E
3D64	00	3D6C	06
3D65	18	3D6D	06
3D66	18	3D6E	00
3D67	00	3D6F	00
3D70	18	3D78	18
3D71	00	3D79	00
3D72	18	3D7A	18
3D73	30	3D7B	18
3D74	66	3D7C	18
3D75	66	3D7D	18
3D76	3C	3D7E	18
3D77	00	3D7F	00
3D80	00	3D88	7C
3D81	00	3D89	C6
3D82	73	3D8A	C6
3D83	DE	3D8B	FC
3D84	CC	3D8C	C6
3D85	DE	3D8D	C6
3D86	73	3D8E	F8
3D87	00	3D8F	C0
3D90	00	3D98	3C
3D91	66	3D99	60
3D92	66	3D9A	60
3D93	3C	3D9B	3C
3D94	66	3D9C	66
3D95	66	3D9D	66
3D96	3C	3D9E	3C
3D97	00	3D9F	00

CHARACTERS

Address	Value
3DA0	00
3DA1	00
3DA2	1E
3DA3	30
3DA4	7C
3DA5	30
3DA6	1E
3DA7	00
3DA8	38
3DA9	6C
3DAA	C6
3DAB	FE
3DAC	C6
3DAD	6C
3DAE	38
3DAF	00
3DB0	00
3DB1	C0
3DB2	60
3DB3	30
3DB4	38
3DB5	6C
3DB6	C6
3DB7	00
3DB8	00
3DB9	00
3DBA	66
3DBB	66
3DBC	66
3DBD	7C
3DBE	60
3DBF	60
3DC0	00
3DC1	00
3DC2	00
3DC3	FE
3DC4	6C
3DC5	6C
3DC6	6C
3DC7	00
3DC8	00
3DC9	00
3DCA	00
3DCB	7E
3DCC	D8
3DCD	D8
3DCE	70
3DCF	00
3DD0	03
3DD1	06
3DD2	0C
3DD3	3C
3DD4	66
3DD5	3C
3DD6	60
3DD7	C0
3DD8	03
3DD9	06
3DDA	0C
3DDB	66
3DDC	66
3DDD	3C
3DDE	60
3DDF	C0
3DE0	00
3DE1	E6
3DE2	3C
3DE3	18
3DE4	38
3DE5	6C
3DE6	C7
3DE7	00
3DE8	00
3DE9	00
3DEA	66
3DEB	C3
3DEC	DB
3DED	DB
3DEE	7E
3DEF	00
3DF0	FE
3DF1	C6
3DF2	60
3DF3	30
3DF4	60
3DF5	C6
3DF6	FE
3DF7	00
3DF8	00
3DF9	7C
3DFA	C6
3DFB	C6
3DFC	C6
3DFD	6C
3DFE	EE
3DFF	00

CHARACTERS

Address	Value
3E00	18
3E01	30
3E02	60
3E03	C0
3E04	80
3E05	00
3E06	00
3E07	00

Address	Value
3E08	18
3E09	0C
3E0A	06
3E0B	03
3E0C	01
3E0D	00
3E0E	00
3E0F	00

Address	Value
3E10	00
3E11	00
3E12	00
3E13	01
3E14	03
3E15	06
3E16	0C
3E17	18

Address	Value
3E18	00
3E19	00
3E1A	00
3E1B	80
3E1C	C0
3E1D	60
3E1E	30
3E1F	18

Address	Value
3E20	18
3E21	3C
3E22	66
3E23	C3
3E24	81
3E25	00
3E26	00
3E27	00

Address	Value
3E28	18
3E29	0C
3E2A	06
3E2B	03
3E2C	03
3E2D	06
3E2E	0C
3E2F	18

Address	Value
3E30	00
3E31	00
3E32	00
3E33	81
3E34	C3
3E35	66
3E36	3C
3E37	18

Address	Value
3E38	18
3E39	30
3E3A	60
3E3B	C0
3E3C	C0
3E3D	60
3E3E	30
3E3F	18

Address	Value
3E40	18
3E41	30
3E42	60
3E43	C1
3E44	83
3E45	06
3E46	0C
3E47	18

Address	Value
3E48	18
3E49	0C
3E4A	06
3E4B	83
3E4C	C1
3E4D	60
3E4E	30
3E4F	18

Address	Value
3E50	18
3E51	3C
3E52	66
3E53	C3
3E54	C3
3E55	66
3E56	3C
3E57	18

Address	Value
3E58	C3
3E59	E7
3E5A	7E
3E5B	3C
3E5C	3C
3E5D	7E
3E5E	E7
3E5F	C3

Address	Data	Address	Data
3E60	03	3E68	C0
3E61	07	3E69	E0
3E62	0E	3E6A	70
3E63	1C	3E6B	38
3E64	38	3E6C	1C
3E65	70	3E6D	0E
3E66	E0	3E6E	07
3E67	C0	3E6F	03
3E70	CC	3E78	AA
3E71	CC	3E79	55
3E72	33	3E7A	AA
3E73	33	3E7B	55
3E74	CC	3E7C	AA
3E75	CC	3E7D	55
3E76	33	3E7E	AA
3E77	33	3E7F	55
3E80	FF	3E88	03
3E81	FF	3E89	03
3E82	00	3E8A	03
3E83	00	3E8B	03
3E84	00	3E8C	03
3E85	00	3E8D	03
3E86	00	3E8E	03
3E87	00	3E8F	03
3E90	00	3E98	C0
3E91	00	3E99	C0
3E92	00	3E9A	C0
3E93	00	3E9B	C0
3E94	00	3E9C	C0
3E95	00	3E9D	C0
3E96	FF	3E9E	C0
3E97	FF	3E9F	C0
3EA0	FF	3EA8	FF
3EA1	FE	3EA9	7F
3EA2	FC	3EAA	3F
3EA3	F8	3EAB	1F
3EA4	F0	3EAC	0F
3EA5	E0	3EAD	07
3EA6	C0	3EAE	03
3EA7	80	3EAF	01
3EB0	01	3EB8	80
3EB1	03	3EB9	C0
3EB2	07	3EBA	E0
3EB3	0F	3EBB	F0
3EB4	1F	3EBC	F8
3EB5	3F	3EBD	FC
3EB6	7F	3EBE	FE
3EB7	FF	3EBF	FF

CHARACTERS

Address	Value
3EC0	AA
3EC1	55
3EC2	AA
3EC3	55
3EC4	00
3EC5	00
3EC6	00
3EC7	00
3EC8	0A
3EC9	05
3ECA	0A
3ECB	05
3ECC	0A
3ECD	05
3ECE	0A
3ECF	05
3ED0	00
3ED1	00
3ED2	00
3ED3	00
3ED4	AA
3ED5	55
3ED6	AA
3ED7	55
3ED8	A0
3ED9	50
3EDA	A0
3EDB	50
3EDC	A0
3EDD	50
3EDE	A0
3EDF	50
3EE0	AA
3EE1	54
3EE2	A8
3EE3	50
3EE4	A0
3EE5	40
3EE6	80
3EE7	00
3EE8	AA
3EE9	55
3EEA	2A
3EEB	15
3EEC	0A
3EED	05
3EEE	02
3EEF	01
3EF0	01
3EF1	02
3EF2	05
3EF3	0A
3EF4	15
3EF5	2A
3EF6	55
3EF7	AA
3EF8	00
3EF9	80
3EFA	40
3EFB	A0
3EFC	50
3EFD	A8
3EFE	54
3EFF	AA
3F00	7E
3F01	FF
3F02	99
3F03	FF
3F04	BD
3F05	C3
3F06	FF
3F07	7E
3F08	7E
3F09	FF
3F0A	99
3F0B	FF
3F0C	C3
3F0D	BD
3F0E	FF
3F0F	7E
3F10	38
3F11	38
3F12	FE
3F13	FE
3F14	FE
3F15	10
3F16	38
3F17	00
3F18	10
3F19	38
3F1A	7C
3F1B	FE
3F1C	7C
3F1D	38
3F1E	10
3F1F	00

Address	Value	Address	Value
3F20	6C	3F28	10
3F21	FE	3F29	38
3F22	FE	3F2A	7C
3F23	FE	3F2B	FE
3F24	7C	3F2C	FE
3F25	38	3F2D	10
3F26	10	3F2E	38
3F27	00	3F2F	00
3F30	00	3F38	00
3F31	3C	3F39	3C
3F32	66	3F3A	7E
3F33	C3	3F3B	FF
3F34	C3	3F3C	FF
3F35	66	3F3D	7E
3F36	3C	3F3E	3C
3F37	00	3F3F	00
3F40	00	3F48	00
3F41	7E	3F49	7E
3F42	66	3F4A	7E
3F43	66	3F4B	7E
3F44	66	3F4C	7E
3F45	66	3F4D	7E
3F46	7E	3F4E	7E
3F47	00	3F4F	00
3F50	0F	3F58	3C
3F51	07	3F59	66
3F52	0D	3F5A	66
3F53	78	3F5B	66
3F54	CC	3F5C	3C
3F55	CC	3F5D	18
3F56	CC	3F5E	7E
3F57	78	3F5F	18
3F60	0C	3F68	18
3F61	0C	3F69	1C
3F62	0C	3F6A	1E
3F63	0C	3F6B	1B
3F64	0C	3F6C	18
3F65	3C	3F6D	78
3F66	7C	3F6E	F8
3F67	38	3F6F	70
3F70	99	3F78	10
3F71	5A	3F79	38
3F72	24	3F7A	38
3F73	C3	3F7B	38
3F74	C3	3F7C	38
3F75	24	3F7D	38
3F76	5A	3F7E	7C
3F77	99	3F7F	D6

CHARACTERS

Address	Value	Address	Value
3F80	18	3F88	18
3F81	3C	3F89	18
3F82	7E	3F8A	18
3F83	FF	3F8B	18
3F84	18	3F8C	FF
3F85	18	3F8D	7E
3F86	18	3F8E	3C
3F87	18	3F8F	18
3F90	10	3F98	08
3F91	30	3F99	0C
3F92	70	3F9A	0E
3F93	FF	3F9B	FF
3F94	FF	3F9C	FF
3F95	70	3F9D	0E
3F96	30	3F9E	0C
3F97	10	3F9F	08
3FA0	00	3FA8	00
3FA1	00	3FA9	00
3FA2	18	3FAA	FF
3FA3	3C	3FAB	FF
3FA4	7E	3FAC	7E
3FA5	FF	3FAD	3C
3FA6	FF	3FAE	18
3FA7	00	3FAF	00
3FB0	80	3FB8	02
3FB1	E0	3FB9	0E
3FB2	F8	3FBA	3E
3FB3	FE	3FBB	FE
3FB4	F8	3FBC	3E
3FB5	E0	3FBD	0E
3FB6	80	3FBE	02
3FB7	00	3FBF	00
3FC0	38	3FC8	38
3FC1	38	3FC9	38
3FC2	92	3FCA	10
3FC3	7C	3FCB	FE
3FC4	10	3FCC	10
3FC5	28	3FCD	28
3FC6	28	3FCE	44
3FC7	28	3FCF	82
3FD0	38	3FD8	38
3FD1	38	3FD9	38
3FD2	12	3FDA	90
3FD3	7C	3FDB	7C
3FD4	90	3FDC	12
3FD5	28	3FDD	28
3FD6	24	3FDE	48
3FD7	22	3FDF	88

CHARACTERS

Address	Value
3FE0	00
3FE1	3C
3FE2	18
3FE3	3C
3FE4	3C
3FE5	3C
3FE6	18
3FE7	00

Address	Value
3FE8	3C
3FE9	FF
3FEA	FF
3FEB	18
3FEC	0C
3FED	18
3FEE	30
3FEF	18

Address	Value
3FF0	18
3FF1	3C
3FF2	7E
3FF3	18
3FF4	18
3FF5	7E
3FF6	3C
3FF7	18

Address	Value
3FF8	00
3FF9	24
3FFA	66
3FFB	FF
3FFC	66
3FFD	24
3FFE	00
3FFF	00

3 LE BASIC

3.1 L'interpréteur Basic du CPC

Le CPC dispose d'un interpréteur Basic rapide et puissant qui est logé dans une Rom de 16 K. Il occupe la zone d'adresses &C000 à &FFFF, parallèlement à la Ram écran. Pour le programme Basic et pour les variables Basic, la zone de &0170 à &A67B est disponible, ce qui représente 42249 octets.

L'interpréteur soutient presque toutes les possibilités offertes par l'électronique et le système d'exploitation de l'ordinateur. Cela comprend notamment la sortie sur écran avec jusqu'à 8 fenêtres, le graphisme haute résolution, le son ainsi que le traitement d'event. Il est ainsi pour la première fois possible de faire exécuter en Basic plusieurs tâches parallèlement. L'interpréteur Basic offre en outre une arithmétique avec des nombres entiers de 16 bits (zone de valeurs de -32768 à 32767) et une arithmétique avec virgule flottante avec un exposant-puissance de deux sur 8 bits et une mantisse de 32 bits qui garantit une précision de 9 décimales pour une zone de valeurs de +- 1E-39 à +- 1E+38.

L'arithmétique entière ou l'arithmétique à virgule flottante ne font cependant pas partie de l'interpréteur Basic mais de la Rom du système d'exploitation (adresses &2F73 à &37FF). Elles sont appelées comme les autres fonctions du système d'exploitation à travers la table de saut qui se trouve dans le haut de la Ram (&BB00 à &BDF4) et qui peut être modifiée en cas de besoin.

L'interpréteur Basic permet également la création, l'édition (examen/modification) et l'exécution aisées de programmes. La création de programmes est en effet facilitée par l'instruction AUTO, l'édition par l'instruction EDIT qui, grâce à la puissance du système d'exploitation, est à peine moins maniable que l'éditeur plein écran, ainsi que par les instructions RENUM, MERGE et DELETE. L'exécution des programmes est également facilitée par des instructions puissantes. Par exemple l'instruction ON ERROR GOTO permet le traitement des erreurs.

L'instruction DEFtype permet de définir le type d'une variable, l'instruction ERASE permet une suppression sélective de tableaux. Il est encore possible d'entrer et de faire sortir les nombres comme des nombres décimaux, binaires ou hexadécimaux ainsi que d'utiliser des fonctions qu'on a soi-même définies, fonctions qui peuvent comporter plusieurs arguments. Enfin les structures de programme telles que IF ... THEN ... ELSE, FOR ... NEXT et WHILE ... WEND sont un autre aspect très important de la puissance du Basic du CPC. Il est également possible en Basic de réaffecter les touches du clavier, de définir les fonctions des touches de fonction ou de définir des caractères qui apparaîtront à l'écran. Il ne manque ni l'instruction TRACE ni une très complète instruction PRINT USING.

Après ce bref aperçu, nous allons nous pencher de plus près sur l'entrée et le stockage des lignes de Basic, ainsi que sur l'exécution des programmes par l'interpréteur Basic. Ces informations vous permettront non seulement de pouvoir tirer le maximum de votre interpréteur Basic mais également d'écrire vos propres extensions du Basic. Nous vous donnerons plus loin quelques exemples d'extensions du Basic.

L'entrée de lignes Basic

Lorsque vous entrez une ligne Basic, elle est d'abord placée dans un buffer de 256 octets qui se trouve aux adresses &ACA8 à &ADA7. L'entrée y figure en clair, non codée. Si la ligne commence par un numéro, celui-ci est converti en un nombre binaire de 16 bits et placé dans un second buffer destiné à recevoir la ligne traitée. Ce buffer comprend 300 caractères et il se trouve avant le programme Basic, aux adresses &40 à &16F. La ligne entrée est alors examinée pour voir si elle comporte des mots-clé Basic. Ces mots-clés sont remplacés par un octet appelé token. Par exemple 'AFTER' devient le token &80. Les tokens de tous les mots d'instruction et des opérateurs Basic tels que '=' ou 'AND' ont des valeurs supérieures à 127, c'est-à-dire que leur bit 7 est mis. Les fonctions Basic comme EXP ou ROUND ont des tokens compris entre 0 et &7F.

Pour les distinguer des caractères ASCII normaux, ils sont marqués par un &FF les précédant. Le double-point servant à séparer entre elles deux instructions est représenté par le code &01, la fin d'une ligne est marquée par un &00. Si une suite de lettres n'a pu être identifiée comme étant une instruction ou une fonction, elle est traitée comme étant le nom d'une variable. Un nom de variable peut comprendre jusqu'à 40 caractères qui sont tous significatifs. Aucune différence n'est faite entre les majuscules et les minuscules. Supposons que nous ayons entré la ligne suivante:

10 start=77

Après le numéro de ligne seront placées les valeurs:

&0D &00 &00 &73 &74 &61 &72 &F4 &EF &19 &4D &00

Le &0D indique qu'il s'agit d'une variable sans marque de type. Ensuite viennent deux 0 sur lesquels nous reviendrons plus tard. Puis vient le nom de la variable, les codes ASCII pour s, t, a et r. Pour la dernière lettre, 't', &80 est ajouté au code ASCII &74 (le bit supérieur est mis) et nous obtenons &F4. Le code &EF est le token pour '='. Le code &19 qui suit indique une constante à un octet: &4D est la valeur de cette constante (=77 en décimal). Le zéro qui termine marque la fin de la ligne.

Avant le numéro de ligne, il y a encore deux octets qui indiquent la longueur de la ligne:

&12 &00 &0A &00

La ligne comporte donc &12+256*&00 = 18 octets et elle porte le numéro de ligne &0A+256*&00, soit 10.

Vous voyez donc qu'au contraire de ce qui est le cas avec d'autres interpréteurs Basic, les constantes ne sont pas placées dans le texte du programme sous forme de textes ASCII, mais sous la forme de leur traduction binaire. Ceci présente un avantage décisif. La conversion du format ASCII au format binaire prend en effet du temps.

Avec la technique utilisée sur le CPC, cette conversion ne s'effectue qu'une seule fois, lors de l'entrée de la ligne et elle n'a donc pas à être effectuée chaque fois que la ligne est exécutée. Il en découle un gain de vitesse dans l'exécution des programmes qui n'est pas négligeable.

Le CPC connaît d'autre part toute une série de constantes numériques qui sont désignées par un token particulier. Les constantes qui ne comprennent par exemple qu'un seul chiffre, soit les nombres de 0 à 9 sont ainsi codées avec les tokens &0E à &17. Elles n'occupent ainsi qu'un octet dans le texte du programme. Nous avons déjà rencontré le token &19 qui marque les valeurs numériques d'un octet. Pour les valeurs entières sur deux octets, il y a trois tokens différents, suivant que la constante a été entrée sous la forme décimale, binaire ou hexadécimale. La valeur de la constante est toujours stockée de la même façon avec un octet faible et un octet fort.

&1A valeur sur deux octets, décimal
&1B valeur sur deux octets, binaire
&1C valeur sur deux octets, hexadécimal

S'il ne s'agit pas d'un nombre entier ou si sa valeur est supérieure à 32767, le nombre est stocké sous la forme d'une valeur à virgule flottante qui est désignée par le token &1F. Le token est suivi de la valeur à virgule flottante sur 5 octets. Nous reviendrons plus tard sur les valeurs à virgule flottante.

Dans ce contexte, les numéros de ligne ont une situation particulière lorsqu'ils suivent par exemple des instructions telles que GOTO, GOSUB ou RUN. Ils sont également stockés sous la forme binaire, mais ils sont désignés par le token &1E.

Lorsqu'un programme est exécuté et qu'il rencontre par exemple une instruction GOTO, il lit alors le numéro de ligne et il doit rechercher cette ligne dans tout le programme. Sur des programmes de taille importante, cela peut durer assez longtemps. Les instructions GOTO et GOSUB sont souvent utilisées dans des boucles qui sont parcourues des centaines ou des milliers de fois.

Dans ce cas, le temps de recherche des numéros de ligne peut représenter un part importante du temps d'exécution du programme. L'interpréteur Basic du CPC n'effectue cette recherche de ligne qu'une seule fois. En effet, une fois qu'il a trouvé la ligne recherchée, il remplace le numéro de ligne figurant à la suite de l'instruction GOTO par l'adresse de cette ligne qu'il vient de trouver. Pour qu'il puisse faire la différence entre un numéro de ligne et une adresse de ligne, il remplace le token &1E par le token &1D, qui est le token pour les adresses de ligne. Si la même instruction GOTO est exécutée une seconde fois, l'interpréteur trouve directement l'adresse à laquelle le programme doit sauter, ce qui permet bien sûr de gagner beaucoup de temps.

Cette technique crée cependant quelques difficultés pour les instructions qui utilisent le numéro de ligne en tant que tel. Lorsque l'instruction LIST doit par exemple sortir le numéro de ligne, c'est le numéro de ligne qu'elle doit indiquer et non l'adresse de la ligne. Ce problème est cependant très facilement résolu. En effet lorsque l'adresse de la ligne est connue, il est facile d'aller y rechercher le numéro de ligne puisque, comme nous l'avons vu, le numéro de ligne est stocké dans la ligne. Lorsque des lignes sont supprimées ou que d'autres lignes sont ajoutées, les adresses de ligne doivent être remplacées par les numéros de ligne car de telles opérations entraînent bien sûr une modification des adresses de ligne. Cela ne présente cependant d'inconvénient que pour l'entrée et la sortie de lignes de programmes. Ce petit inconvénient est cependant largement compensé par la vitesse nettement plus grande d'exécution des programmes.

L'exécution des programmes par l'interpréteur Basic

L'exécution d'une instruction par l'interpréteur Basic se présente, en simplifiant un peu, de la façon suivante. Chaque ligne de programme commence, comme nous l'avons dit, par la longueur de la ligne et le numéro de ligne. Ensuite vient l'instruction Basic proprement dite. L'interpréteur examine maintenant s'il s'agit d'un token d'instruction, dont la valeur est toujours comprise entre &80 et &DC.

Si c'est le cas, il utilise ce token comme pointeur d'une table qui contient les adresses de toutes les instructions Basic. L'instruction Basic est alors exécutée comme un sous-programme. On revient ensuite à ce qu'on appelle la boucle de l'interpréteur. Si l'instruction ne commençait cependant pas par un token d'instruction, on saute à l'instruction LET.

La partie la plus importante de l'interpréteur Basic est certainement le calcul des expressions. Le CPC distingue à cet égard trois types d'expressions: entières, à virgule flottante et chaînes de caractères. Lorsque par exemple une affectation de valeur à une variable est exécutée ou lorsque le paramètre d'une instruction doit être calculé, une routine est appelée qui calcule l'expression et qui fournit la valeur ainsi que le type de l'expression. Le type de variable peut avoir trois valeurs différentes:

2 entier
3 chaîne
5 virgule flottante

Ce numéro de type donne en même temps la longueur de la variable. Pour une chaîne, c'est ce qu'on appelle le Descriptor qui contient la longueur et l'adresse de la chaîne (voyez également le chapitre sur le pointeur de variable). Si cependant le type d'une expression est différent du type d'une variable à laquelle cette expression doit être affectée, une conversion de type est tentée, mais seulement entre les deux types numériques entier et à virgule flottante. Cette conversion prend bien sûr un certain temps et il est donc préférable d'employer des variables entières lorsque c'est possible. L'expérience révèle en effet que le type entier convient dans 90 % des cas. Non seulement le type entier évite les conversions de types, mais l'arithmétique entière est en outre nettement plus rapide que l'arithmétique à virgule flottante. Cette remarque vaut particuliérement pour les variables de comptage utilisées par exemple dans les boucles FOR...NEXT.

Par contre, si vous tentez d'affecter une expression du type chaîne de caractères à une variable numérique ou vice versa, le message d'erreur 'Type mismatch' sera sorti. La conversion de chaîne de caractères à numérique et vice versa n'est possible qu'avec les fonctions VAL et STR$.

3.2 La pile Basic

Une pile ou mémoire de pile (stack) permet de stocker des données suivant le principe 'Last in - First out' (dernier entré - premier sorti). Le processeur utilise à cet effet la zone de mémoire commençant en &C000. Avant chaque entrée, le pointeur de pile (stack pointer) est décrémenté. Lorsqu'on retire des données de la pile, le pointeur de pile est incrémenté immédiatement aprés. La pile du processeur sert par exemple à placer les adresses de retour lors de l'appel de sous-programmes et elle permet, grâce au principe d'accès utilisé, de réaliser une imbrication des sous-programmes.

L'interpréteur Basic a également besoin d'une pile pour stocker les paramètres des appels par GOSUB ou des boucles FOR-NEXT et WHILE-WEND. Seule une pile permet en effet de réaliser une imbrication de ces différentes structures de programme. On n'utilise pas à cet effet la pile du processeur car il existe une pile Basic de 512 octets qui commence à l'adresse &AE8B. Au contraire de la pile du processeur, cette pile croît vers les adresses plus élevées, au fur et à mesure que le nombre d'entrées augmente, jusqu'à l'adresse limite &B08A. Les cases mémoire &B08B et &B08C font office de pointeur de pile.

Voyons d'abord quels paramètres sont placés sur la pile pour une instruction GOSUB:

&00/&01 marque du type de GOSUB

Lo   Adresse de l'instruction suivant
Hi   l'instruction GOSUB

Lo   Adresse de la ligne de
Hi   l'instruction GOSUB

&06   Taille de l'entrée sur la pile

Un octet est donc tout d'abord placé sur la pile qui détermine le type de l'instruction GOSUB. Pour un GOSUB normal, il s'agit d'un octet nul. S'il s'agit cependant de l'appel d'un sous-programme par une instruction AFTER ou EVERY, c'est un 1 qui sera placé sur la pile. Viennent ensuite l'adresse de la prochaine instruction après l'instruction GOSUB ainsi que l'adresse de la ligne dans laquelle figure l'instruction GOSUB. Pour que l'entrée sur la pile puisse être identifiée à nouveau lorsque l'instruction RETURN sera exécutée, un octet est encore placé sur la pile qui indique la longueur de l'entrée sur la pile et indique ainsi implicitement qu'il s'agit d'un enregistrement concernant une instruction GOSUB.

Les données pour une boucle WHILE-WEND sont placées de façon similaire:

Lo   Adresse de la ligne de
Hi   l'instruction WHILE

Lo   Adresse de
Hi   l'instruction WEND

Lo   Adresse de
Hi   la condition WHILE

&07   Taille de l'entrée sur la pile

L'entrée comporte donc trois adresses et un octet d'identification qui vaut 7 et qui indique également le nombres d'octets de données entrés sur la pile.

Les choses se compliquent un peu avec la boucle FOR-NEXT. On fait ici une distinction selon que la variable de comptage est du type entier ou du type réel. Dans le premier cas, non seulement le temps d'exécution est plus court, mais la place occupée sur la pile est en outre moindre. Considérons tout d'abord la structure d'une boucle de type entier.

Lo	Adresse de la
Hi	variable de comptage
Lo	Valeur finale de la
Hi	variable de comptage
Lo	Valeur STEP
Hi	
Sgn	Signe de la valeur STEP
Lo	Adresse de
Hi	l'instruction FOR
Lo	Adresse de la ligne de
Hi	l'instruction FOR
Lo	Adresse de
Hi	l'instruction NEXT
Lo	Adresse de la ligne
Hi	de l'instruction NEXT
&10	Taille de l'entrée sur la pile

L'entrée sur la pile pour une boucle FOR-NEXT avec variable entière est donc longue de 16 octets. Si une boucle utilise une variable de comptage de type réel, ce sont 22 octets qui seront placés sur la pile.

Lo Hi	Adresse de la variable de comptage
Valeur à virgule flottante sur 5 octets	Valeur finale de la variable de comptage
Valeur à virgule flottante sur 5 octets	Valeur STEP
Sgn	Signe de la valeur STEP
Lo Hi	Adresse de l'instruction FOR
Lo Hi	Adresse de la ligne de l'instruction FOR
Lo Hi	Adresse de l'instruction NEXT
Lo Hi	Adresse de la ligne de l'instruction NEXT
&16	Taille de l'entrée sur la pile

Outre le stockage des structures de programme, la pile Basic sert également au stockage d'expressions provisoires pour les calculs numériques, par exemple pour le calcul d'expressions imbriquées entre parenthèses et pour réaliser une hiérarchie pour les opérateurs arithmétiques et logiques.

3.3 Basic et langage-machine

3.3.1 L'instruction CALL

L'instruction CALL sert de lien entre le Basic et le langage-machine. Elle permet en effet d'appeler à partir d'un programme Basic un programme en langage-machine. L'instruction CALL doit être accompagnée d'une adresse 16 bits qui indique en quelle adresse figure le programme en langage-machine, par exemple:

CALL &8000

Cette instruction appellera un programme en langage-machine figurant à l'adresse &8000 ou 32768 en décimal. Si le programme en langage-machine se termine par une instruction RET, le contrôle est rendu à l'interpréteur qui poursuit l'exécution du programme Basic.

Avec l'instruction CALL on ne peut accéder directement au système d'exploitation ou à l'interpréteur Basic. Pour toute la zone d'adresses de 64 K, c'est la Ram qui est sélectionnée automatiquement. Il est cependant possible évidemment d'appeler des routines du système d'exploitation à travers les adresses d'entrée qui figurent en &B000. Ces routines s'occupent elles-mêmes de réaliser la configuration Rom/Ram qui convient. Si vous voulez accéder avec une instruction CALL à des routines de l'interpréteur Basic ou à des routines du système d'exploitation qui ne peuvent être appelées avec des vecteurs, vous pouvez utiliser les routines RST 3 et RST 5 qui réalisent la commutation.

L'instruction CALL permet cependant également de transmettre des paramètres du Basic à la routine en langage-machine. Vous pouvez pour cela transmettre jusqu'à 32 paramètres qui doivent être placés à la suite de l'instruction CALL, séparés par des virgules. Ces paramètres, ainsi que l'adresse elle-même doivent donner une valeur 16 bits. Ils sont placés par le Basic sur la pile. L'interpréteur Basic transmet l'adresse de base du bloc de paramètres dans le registre IX. Dans l'accumulateur figure le nombre de paramètres transmis.

Le dernier paramètre figure donc à l'adresse IX, l'avant-dernier à l'adresse IX+2 et le premier paramètre à l'adresse IX+2*(A-1).

Pendant l'instruction CALL, les contenus de tous les registres peuvent être modifiés. Le pointeur de pile peut lui aussi être modifié pour autant qu'on soit sûr que lors de l'exécution de l'instruction RET qui termine le programme en langage-machine, c'est bien la bonne adresse de retour qui sera retirée de la pile.

Les applications possibles de l'instruction CALL sont très diverses et vous pouvez dans ce domaine donner libre cours à votre imagination. Vous pouvez par exemple créer des fonctions graphiques nouvelles telles que le dessin de cercles, le remplissage de surfaces, etc...

La transmission de paramètres en retour, de la routine en langage-machine au Basic n'est pas prévue mais elle reste cependant possible par un petit détour. Si par exemple le résultat d'un programme en langage-machine doit être affecté à une variable, on peut transmettre l'adresse de cette variable à travers l'instruction CALL, grâce au signe 'arobas':

```
CALL &AB00,@A
```

L'adresse de la variable A sera ainsi à la disposition du programme en langage-machine qui pourra modifier directement la valeur de cette variable. Cette possibilité est décrite plus précisément dans le chapitre sur le pointeur de variable.

3.3.2 Extensions du Basic avec RSX

Le système d'exploitation et le Basic du CPC soutiennent la possibilité d'intégrer ses propres instructions dans le Basic. C'est ce qu'on appelle RSX 'Resident System eXtension'. Ces extensions peuvent être appelées en Basic à travers un nom, et elles permettent une transmission de paramètres comme nous l'avons déjà décrite pour l'instruction CALL.

Si nous voulons par exemple écrire une extension graphique qui dessine un carré sur l'écran, l'appel de cette fonction se présentera ainsi:

IQUADRAT,100,100,50

Nous voulons ainsi dessiner un carré dont l'angle supérieur gauche aura les coordonnées 100, 100 avec un côté d'une longueur de 50 points.

Comme vous voyez, une extension d'instruction est marquée par un trait vertical (SHIFT @) placé devant le mot instruction.

Une telle extension d'instruction peut figurer dans une Rom d'extension, comme celle par exemple qui gère le lecteur de disquette, ou bien également en Ram. Cela nous donne donc la possibilité d'écrire nos propres extensions d'instruction. Pour que le système d'exploitation sache où il doit chercher une telle extension, l'extension doit d'abord être 'intégrée'. On emploie pour cela une routine du système d'exploitation: KL LOG EXT. L'exemple suivant réalise l'instruction évoquée ci-dessus pour dessiner un carré et montre comment l'intégration se réalise.

```
;RSX - EXTENSIONS D'INSTRUCTION
                           ;L.E. 15/6/85

BCD1                       LOGEXT   EQU   &BCD1 ; intégrer extension
BBC6                       ASKCUR   EQU   &BBC6 ; amener cursr graphique
BBC0                       MOVABS   EQU   &BBC0 ;fixer curseur graphique
BBF9                       DRAWRE   EQU   &BBF9 ;tracer ligne relativ.
BDC7                       CHGSGN   EQU   &BDC7 ;modifier signe

8000                                ORG   &8000

8000      010980                    LD    BC,RSX ;adresse table instr.RSX
8003      211680                    LD    HL,KERNAL ;4 oct.Ram pour Kernl
8006      C3D1BC                    JP    LOGEXT ; intégrer extension
```

```
8009    0E80       RSX      DEFW TABLE ;Adresse des mots instr.
800B    C31A80              JP   QUADRAT
800E    51554144   TABLE    DEFM "QUADRA"
8014    D4                  DEFB "T"+&80
8015    00                  DEFB 0 ; fin de la table

8016               KERNAL   DEFS 4 ; mémoire pour Kernal

801A    FE03       QUADRA   CP   3 ; trois paramètres?
801C    C0                  RET  NZ
801D    CDC6BB              CALL ASKCURS ;amener cursr graphique
8020    D5                  PUSH DE ; ranger coordonnée X
8021    E5                  PUSH HL ; ranger coordonnée Y
8022    DD5605              LD   D,(IX+5)
8025    DD5E04              LD   E,(IX+4)  ;coordonnée X
8028    DD6603              LD   H,(IX+3)
802B    DD6E02              LD   L,(IX+2)  ;coordonnée Y
802E    CDC0BB              CALL MOVABS ;CursrGraph.surCoord.XY
8031    DD5601              LD   D,(IX+1)
8034    DD5E00              LD   E,(IX) ;ranger longueur dans de
8037    D5                  PUSH DE ;comme offset X
8038    210000              LD   HL,0 ;offset Y
803B    CDF9BB              CALL DRAWREL ;tracer ligne horiz.
803E    E1                  POP  HL
803F    E5                  PUSH HL
8040    CDC7BD              CALL CHGSGN ;offset Y négatif
8043    E5                  PUSH HL
8044    110000              LD   DE,0
8047    CDF9BB              CALL DRAWREL ;tracer ligne verticale
804A    D1                  POP  DE ;offset X négatif
804B    210000              LD   HL,0 ;offset Y nul
804E    CDF9BB              CALL DRAWREL ;tracer ligne horiz.
8051    E1                  POP  HL
8052    110000              LD   DE,0
8055    CDF9BB              CALL DRAWREL ;tracer ligne verticale
8058    E1                  POP  HL
8059    D1                  POP  DE
805A    C3C0BB              JP   MOVABS ;rétablir coordonnées
```

Après que ce programme ait été chargé (comme fichier binaire à partir de la disquette) ou qu'il ait été placé en mémoire avec un programme de chargement de DATA, il doit être initialisé une seule fois. Il faut pour cela utiliser l'appel CALL &8000. La nouvelle instruction est alors disponible. Deux tables sont utilisées pour l'intégration. La première, appelée RSX dans notre exemple, contient tout d'abord l'adresse de la seconde table, appelée ici TABLE, suivie des instructions de saut à l'extension proprement dite.

La seconde table contient les noms sous lesquels les nouvelles instructions peuvent être appelées. Les majuscules et les points sont autorisés. Le dernier caractère d'un mot instruction est marqué par son bit 7 qui est mis. La fin de la table est indiquée par un octet nul. Chaque table doit bien sûr contenir le même nombre d'entrées. Pour chaque mot d'instruction doit figurer l'adresse de saut correspondante dans la première table. Sous l'étiquette KERNAL, nous devons mettre 4 octets à la disposition du système d'exploitation qui sont utilisés pour la gestion de l'extension. Les 4 octets doivent être placés entre l'adresse &4000 et l'adresse &BFFF.

La routine de dessin d'un carré commence par l'étiquette QUADRAT (quadrate en anglais=carré). On contrôle d'abord si trois paramètres ont bien été transmis. Si ce n'est pas le cas, on quitte la routine immédiatement. Mais si c'est le cas, on va chercher la position actuelle du curseur graphique et on la range sur la pile. On va ensuite chercher dans de et hl les coordonnées X et Y transmises. La base du bloc de paramètres se trouve en IX. Après que le curseur graphique ait été fixé sur ces coordonnées, la routine de dessin d'une ligne relativement à la position actuelle peut être appelée quatre fois. Pour calculer un offset négatif, on appelle la routine CHGSGN de l'arithmétique entière. Pour finir, on rétablit la position originelle du curseur.

Voici un exemple d'utilisation de cette routine:

```
10 CLS
20 FOR i=35 TO 400 STEP 20
30 IQUADRAT,i,i,30
40 NEXT
```

3.3.3 Le pointeur de variable '@'

Une fonction particulièrement intéressante pour le programmeur en langage-machine est constituée par le pointeur de variable qui est appelé avec l'arobas. Cette fonction renvoie l'adresse où est placée une variable. L'appel de cette fonction se présente ainsi:

```
PRINT @a
```

On sort ainsi l'adresse de la variable a. Si la variable n'avait pas encore été initialisée, le message d'erreur 'Improper argument' sera sorti.

Si nous voulons maintenant accéder au contenu de la variable, nous devons distinguer entre les 3 différents types possibles.

La situation est très simple en ce qui concerne les variables entières. La valeur 16 bits est placée à l'adresse fournie. Nous pouvons donc obtenir la valeur de la variable a% avec la formule:

```
PRINT PEEK(@a%)+256*PEEK(@a%+1)
```

Nous pouvons ainsi obtenir des valeurs entre 0 et 65535. Si nous voulons tenir également compte du signe, nous devons utiliser la fonction UNT.

```
PRINT UNT(PEEK(@a%)+256*PEEK(@a%+1))
```

Pour les variables à virgule flottante, le pointeur de variable est également dirigé sur la valeur de la variable, mais celle-ci

est exprimée avec 5 octets.
Les 5 premiers octets sont ce qu'on appelle la mantisse et le cinquième octet est la puissance de 2 par laquelle doit être multipliée la mantisse pour obtenir la valeur de la variable. Si nous désignons les 4 octets de la mantisse par m1 à m4 et l'exposant par ex, nous obtenons la valeur à virgule flottante avec la formule suivante:

```
x=(1-2*SGN(m4 AND 128))*2^(ex-129)*
  (1+((m4 AND 127)+(m3+(m2+m1/256)/256)/256)/128)
```

La formule met en évidence que le signe du nombre à virgule flottante se trouve dans le bit supérieur de m4 et que les octets de la mantisse m1 à m4 ont des valeurs croissantes. La puissance de 2 contient un offset de 129 ce qui donne des valeurs de 2^-129 à 2^127. Essayons notre formule:

```
100 a=-13:'variable a virgule flottante examinee
110 ad=@a:'adresse de a
120 m1=PEEK(ad):m2=PEEK(ad+1):m3=PEEK(ad+2)
130 m4=PEEK(ad+3):ex=PEEK(ad+4)
140 PRINT(1-2*SGN(m4 AND 128))*2^(ex-129)*
    (1+((m4AND127)+(m3+(m2+m1/256)/256)/256)/128)
```

Si vous faites tourner ce programme, vous obtiendrez en résultat la valeur -13. Remplacez si vous le voulez la ligne 100 par INPUT a et vous pourrez tester n'importe quelles valeurs.

La fonction de pointeur de variable trouve son application dans l'instruction CALL qui ne peut en effet transmettre que des valeurs 16 bits. Si vous voulez donc travailler avec des valeurs à virgule flottante, vous pouvez transmettre avec '@' l'adresse d'une variable à virgule flottante. Vous pourrez ensuite vous référer à cette adresse. Cette méthode permet également bien sûr de modifier directement la valeur d'une variable à virgule flottante.

Le cas des variables alphanumériques est encore plus intéressant. Ici aussi, nous pouvons utiliser le pointeur de variable qui nous

renvoie l'adresse de la variable.
Ce n'est cependant pas directement l'adresse de la chaîne de caractéres mais celle de ce qu'on appelle le descripteur de chaîne. Ce descripteur de chaîne est long de trois octets. Le premier octet contient la longueur de la chaîne, soit une valeur entre 0 et 255. Les deux octets suivants contiennent l'adresse de la chaîne.

```
100 INPUT a$
110 ad=@a$
120 I=PEEK(ad)
130 sa=PEEK(ad+1)+256*PEEK(ad+2)
140 FOR i=sa TO sa+I-1:PRINT CHR$(PEEK(I));:NEXT
```

Ce programme va chercher la longueur et l'adresse de la chaîne, la lit et la sort.

Ici aussi, il est possible de transmettre une chaîne à l'instruction CALL à travers le pointeur de variable.

Les chaînes peuvent être encore employées en liaison avec l'instruction CALL de façon tout à fait différente. On peut par exemple placer tout simplement un programme en langage-machine dans une chaîne et l'appeler avec l'instruction CALL et le pointeur de variable. Le programme en langage-machine doit pour cela être transposable (il ne doit pas contenir d'adresse absolue interne) et il ne doit pas comporter plus de 255 octets. La plupart des petits programmes utilitaires remplissent ces conditions. Si vous voulez utiliser cette méthode, il vous faut procéder ainsi:

Le programme en langage-machine est d'abord placé dans la variable alphanumérique. On utilisera le plus souvent READ et DATA à cet effet. Si vous voulez ensuite faire exécuter le programme, il vous suffit de faire calculer l'adresse de début de la chaîne de caractéres (et donc du programme) avec l'arobas.

3.4 La Rom Basic

3.4.1 L'arithmétique à virgule flottante

Toutes les fonctions arithmétiques qu'utilise l'interpréteur Basic se trouvent dans la Rom du système d'exploitation. Elles sont appelées à travers une table de saut placée en &BD5E à &BDBB. Si vous voulez modifier les routines arithmétiques, il vous suffit d'insérer à l'emplacement voulu un saut à votre routine.

Nous allons vous montrer comme exemple d'application des routines avec virgule flottante une routine de calcul de la racine carrée d'un nombre. L'interpréteur Basic du CPC nous fournit certes déjà cette fonction mais nous voulons démontrer que celle-ci peut être encore améliorée par l'emploi d'algorithmes plus puissants.

La fonction SQR intégrée travaille d'après le même algorithme que le calcul de la puissance.

SQR(X)=EXP(LOG(X)*0.5)

Il faut donc calculer chaque fois les fonctions exponentielle et logarithme, ce qui s'effectue à travers des calculs de polynômes compliqués et longs. La racine carrée peut cependant être calculée simplement à travers un processus d'itération.

X(N+1)=(X(N)+A/X(N))/2

où A est le nombre dont la racine doit être extraite, X(N) est l'ancienne et X(N+1) la nouvelle valeur approchée. Comme valeur de départ, on peut prendre le nombre A lui-même. On obtient une meilleure valeur approchée lorsqu'on divise par deux la puissance de deux du nombre à virgule flottante. Le résultat ne se modifie plus ensuite, après 4 itérations, dans le cadre de la précision de calcul. Notez également que la division par deux n'a pas été réalisée avec une division à virgule flottante qui prend beaucoup de temps. On a simplement décrémenté de 1 la puissance de deux. Le gain de temps dû à ce procédé est significatif.

La routine SQR de l'interpréteur met en effet 27 millisecondes, alors que notre routine exécute la même tâche en 8 millisecondes. Elle est donc plus de trois fois plus rapide.

```
;ROUTINE SQR RAPIDE
                        ;L.E. 10/6/85

A000                            ORG   &A000

BD91              SGN           EQU   &BD91
BD85              DIV           EQU   &BD85
BD79              ADD           EQU   &BD79

A000    CD70BD    NEWSQR        CALL  SGN ;examiner signe
A003    3F                      CCF
A004    C8                      RET   Z ;zéro, déjà terminé
A005    F20CA0                  JP    P,GOON
A008    3E01                    LD    A,1 ;'IMPROPER ARGUMENT'
A00A    B7                      OR    A
A00B    C9                      RET

A00C    E5        GOON          PUSH  HL
A00D    1153A0                  LD    DE,STORE1
A010    010500                  LD    BC,5
A013    EDB0                    LDIR  ;ranger radicande
A015    E1                      POP   HL

A016    E5                      PUSH  HL
A017    DDE1                    POP   IX
A019    DD7E04                  LD    A,(IX+4) ;puissance
A01C    D681                    SUB   &81 ;normaliser
A01E    3F                      CCF
A01F    1F                      RRA   ;diviser puissance par deux
A020    C601                    ADD   A,1
A022    DD7704                  LD    (IX+4),A ;comme valeur départ

A025    0604                    LD    B,4 ;4 itérations
A027    C5        ITER          PUSH  BC
A028    E5                      PUSH  HL
```

```
A029    1158A0            LD    DE,STORE2
A02C    010500            LD    BC,5
A02F    EDB0              LDIR  ;ranger valeur approchée
A031    E1                POP   HL
A032    E5                PUSH  HL
A033    1153A0            LD    DE,STORE1
A036    EB                EX    DE,HL
A037    010500            LD    BC,5
A03A    EDB0              LDIR  ;aller chercher radicande
A03C    E1                POP   HL
A03D    1158A0            LD    DE,STORE2
A040    CD64BD            CALL  DIV
A043    1158A0            LD    DE,STORE2
A046    CD58BD            CALL  ADD
A049    E5                PUSH  HL
A04A    DDE1              POP   IX
A04C    DD3504            DEC   (IX+4) ;nombre/2
A04F    C1                POP   BC
A050    10D5              DJNZ  ITER
A052    C9                RET

A053              STORE1  DEFS  5
A058              STORE2  DEFS  5
```

Mais comment faire pour que l'interpréteur utilise la nouvelle routine? C'est le vecteur &BD9D qui sert pour la fonction SQR. Il faut donc placer en cet endroit un saut à notre routine:

JP &A000

Lorsque la routine est appelée en Basic, le registre HL doit être pointé sur la valeur à virgule flottante. Après exécution de la routine, le registre HL doit être pointé sur le résultat. Normalement la valeur de registre ne doit pas avoir été modifiée. Les flags indiquent l'état des erreurs de la fonction:

Etat des erreurs de la fonction:

C=1	exécution correcte
C=0 & Z=1	'Division by zero'
C=0 & N=1	'Overflow'
C=0 & Z=0	'Improper argument'

Vous trouverez dans les pages suivantes le listing de l'arithmétique à virgule flottante. Chaque routine contient également l'adresse de la table de saut à travers laquelle elle est appelée par l'interpréteur Basic. Vous trouverez ensuite, dans le chapitre consacré à la ROM BASIC, aux adresses DD2F à DE19, l'arithmétique entière qui est utilisée par l'interpréteur chaque fois que c'est possible. En effet comme elle ne travaille qu'avec des valeurs sur deux octets, cette arithmétique est toujours nettement plus rapide que le calcul avec des nombres à virgule flottante. Servez-vous également de ce fait dans vos programmes et utilisez autant que possible des variables entières. Cela vaut notamment pour les boucles FOR-NEXT (voyez également à ce sujet le chapitre 3.2).

****** CPC 664 & 6128 BASIC 1.1
C000 première ROM de premier plan
C001 marque 1
C002 Version 1
C003 Modification 0
C004 Adresse du nom
****** initialisation du BASIC
C006 Stack à partir de C000
C009 KL ROM WALK
C00C configurer la mémoire
C00F trop peu de mémoire, alors Reset
C013 supprimer flag pour inhibation espaces
C016 pointeur sur ' BASIC 1.1'
C019 sortir texte
C01C adresse de ligne actuelle sur zéro
C01F annuler numéro d'erreur
C022 RND-Init
C025 annuler mode AUTO
C028 instruction NEW
C02B 240
C02E SYMBOL AFTER 240
C031 au mode READY
C033 ' BASIC 1.1', LF,LF,0
C040 'BASI', 'C'+80H,0
******* instruction BASIC EDIT
C046 amener numéro de ligne dans DE
C04A initialiser la pile
C04D chercher ligne BASIC DE
C050 lister ligne BASIC dans buffer
C053 aller chercher ligne d'entrée
******* mode READY
C058 initialiser la pile
C05B diverses initialisations
C05E aller chercher adresse de ligne
C061 SOUND HOLD
C064 supprimer Break Event
C067 initialiser l'écran
C06A programme protégé ?
C06E oui, supprimer programme et variables

C071 numéro ERROR
C074 'Syntax error' ?
C076 non
C078 numéro ERROR sur zéro
C07B aller chercher numéro de ligne de la ligne ERROR
C07F à l'instruction EDIT
C081 pointeur sur 'Ready'
C084 sortir
C087 adresse de ligne actuelle sur zéro
C08A AUTO-Flag mis ?
C08E non
C090 indiquer prochain numéro de ligne
C093 au mode READY
C095 ignorer espace, TAB et LF
C09D ignorer espace, TAB et LF
C0AD
C0AF aller chercher ligne d'entrée
C0B2 'ESC' appuyée, alors répéter
C0B4 sortir LF
C0B7 ignorer espace, TAB et LF
C0CB
C0D4 à la boucle de l'interpréteur
C0D7 'Ready', LF,0
******* annuler mode AUTO
C0DF
******* fixer mode AUTO
C0E1 numéro de ligne
C0E6 mettre flag pour AUTO
******* instruction BASIC AUTO
C0EA 10, Default
C0EF ','
C0F1 amener numéro de ligne dans DE
C0F5 10, Default
C0F8 suit virgule ?
C0FB oui, amener numéro de ligne dans DE
C0FE fin de ligne, sinon 'Syntax error'
C102 ranger incrément AUTO
C106 ranger flag pour mode AUTO
C10D numéro de ligne

C115 annuler mode AUTO
C118 éditer ligne
C11E numéro de ligne
C121 plus incrément
C122 fixer mode AUTO
****** instruction BASIC NEW
C128
C129 supprimer programme et variables
C12C au mode READY
****** instruction BASIC CLEAR
C12F 'INPUT'
C13A diverses initialisations
****** CLEAR INPUT
C13F ignorer les espaces
****** supprimer programme et variables
C145 début de la RAM libre
C149 HIMEM
C152 annuler accu
C154 supprimer début RAM libre jusqu'à HIMEM
C156 annuler flag pour programme protégé
C159 restaurer pointeur de variable
C15F interrompre Disk I/O
C163 fixer mode RAD
C166 initialiser pile descripteur
C16C Stream-Reset
C16F TROFF
C172 supprimer mode AUTO
C175 diverses initialisations, voir plus bas
C17A supprimer Strings
C17D restaurer pointeur de variable
C180 toutes les variables sur type 'Real'
C183 ignorer espace, TAB et LF
****** diverses initialisations
C189 initialiser taquets de tabulation
C18C annuler pointeur de programme
C18F annuler ON ERROR
C192 annuler pointeur de programme après interruption
C195 reset SOUND et event
C198 initialiser pile BASIC

C19B annuler flag pour FN
C19E RESTORE
C1AB < 8 ?
C1AD TXT STR SELECT

C1B1 actuel numéro stream
C1B7 canal d'entrée
C1C1 actuel numéro stream
C1C4 imprimante ?
C1C7 canal d'entrée
C1CA disquette ?
C1CD tester sur numéro stream
C1D2 tester numéro stream
C1D7 tester numéro stream
****** aller chercher numéro stream
C1E8 tester numéro stream
C1ED 'Improper argument'
C1F5 ','
C1F7 saut dans (BC), exécuter fonction
******* tester sur numéro stream
C1FF '#'
C201 0 comme défaut
C204 aller chercher numéro stream
C208 suit virgule ?
C20B non, alors fin de l'instruction
****** Streamnummer holen
C210 Test sur nachfolgendes Zeichen
C213 "#"
C214 10, valeur maximum+1
C218 valeur maximum dans B
C219 aller chercher valeur 8 bits
C21C comparer avec valeur maximum
C21F plus petit, ok
C220 'Improper argument'
****** aller chercher valeur 8 bits plus petite que 2
C223 valeur maximum 2
C225 aller chercher argument et tester
****** instruction BASIC PEN
C227 aller chercher numéro stream

C22A TXT SET PEN
C230 aller chercher suit virgule ?
C234 valeur 8 bits plus petit 2
C237 TXT SET BACK
****** instruction BASIC PAPER
C23C aller chercher numéro stream
C23F TXT SET PAPER
C242 aller chercher argument < 16
C246 saut dans (BC), exécuter fonction
****** instruction BASIC BORDER
C24B aller chercher 2 arguments plus petits que 32
C24F SCR SET BORDER
****** instruction BASIC INK
C254 aller chercher argument plus petit que 16
C258 tester si ','
C25B aller chercher 2 arguments plus petits que 32
C260 SCR SET INK
****** aller chercher 2 arguments plus petits que 32
C265 aller chercher argument < 32
C268 dans B
C269 suit virgule ?
C26D 32
C26F aller chercher argument plus petit que 32
C272 dans C
****** aller chercher argument < 16
C274 16
C276 aller chercher argument plus petit que 16
****** instruction BASIC MODE
C278 3
C27A aller chercher argument plus petit que 3
C27E SCR SET MODE
****** instruction BASIC CLS
C283 aller chercher numéro stream
C287 TXT CLEAR WINDOW
C28C aller chercher numéro stream
C291 'Improper argument'
C294 tester si ')'
****** fonction BASIC COPYCHR$
C29B aller chercher numéro stream, parenthèse fermée

C29E TXT RD CHAR
C2A1 accepter caractère comme chaîne
****** fonction BASIC VPOS
C2A4 aller chercher numéro stream
C2A8 aller chercher ligne curseur
****** fonction BASIC POS
C2AD aller chercher numéro stream
C2B0 tester si ')'
C2B4 aller chercher position
C2B7 accepter contenu accu comme nombre entier
****** aller chercher position PRINT actuelle
****** aller chercher ligne curseur
C2CA TXT GET CURSOR
C2CD TXT VALIDATE
C2DA TXT GET WINDOW
****** instruction BASIC LOCATE
C302 aller chercher numéro stream
C305 aller chercher deux valeurs 8 bits différentes de zéro
C30C TXT SET CURSOR
****** instruction BASIC WINDOW
C311 'SWAP'
C315 aller chercher numéro stream
C318 aller chercher deux valeurs 8 bits différentes de zéro
C31C tester si ','
C31F aller chercher deux valeurs 8 bits différentes de zéro
C326 TXT WIN ENABLE

****** WINDOW SWAP
C32B ignorer les espaces
C32E aller chercher argument < 8
C332 suit virgule ?
C335 défaut zéro
C337 oui, aller chercher argument < 8
C33C TXT SWAP STREAMS
****** aller chercher argument < 8
C341 8, valeur maximum
C343 aller chercher argument
****** instruction BASIC TAG
C346 aller chercher numéro stream

****** instruction BASIC TAGOFF
C34D aller chercher numéro stream
C351 TXT SET GRAPHIC
****** aller chercher deux valeurs 8 bits différentes de zéro
C354 aller chercher première valeur
C357 dans D
C358 tester si ','
C35C aller chercher valeur 8 bits différente de zéro
C360 valeur dans E
****** instruction BASIC CURSOR
C363 aller chercher numéro stream
C366 zéro ?
C368 aller chercher valeur 8 bits < 2
C36C TXT CUR OFF
C36F TXT CUR ON
C372 suit virgule ?
C375 non
C376 aller chercher valeur 8 bits < 2
C37A TXT CUR DISABLE
C37D TXT CUR ENABLE
****** sortir chaîne
C380 adresse chaîne
C381 132
C384 WIDTH sur 132
C387 fixer POS sur un
C390 aller chercher caractère
C391 incrémenter pointeur
C392 dernier caractère ?
C393 non, sortir
C396 prochain caractère
****** sortir LF
C39C LF
C39E sortir
****** sortir caractère
C3A5 sortir caractère
C3AB LF
C3AD non
C3AF périphérique de sortie
C3B2 imprimante ?

C3B5 disque ?
C3B8 sortir caractère
****** sortir caractère
C3BE sortir caractère
****** canal de sortie sélectionné
C3C4 canal de sortie
C3C9 sortir sur imprimante
C3CC sur disque
C3D0 sur écran
C3D4 TXT SET GRAPHIC
C3D9 TXT SET BACK
C3DD TXT VDU ENABLE
C3E0 TXT VALIDATE
C3E5 CR
C3EA LF
C3EC TXT OUTPUT
C3F1 TXT VALIDATE
****** sortir CR & LF sur imprimante
C3F8 CR
C3FD LF
C40B MC PRINT CHAR
C40F tester si interruption avec 'ESC'
C415 CR
C41A ' '
C434 CR
C439 LF
C449 DISK OUT CHAR
C44C pas d'erreur ?
C44D sortir message d'erreur
C44F fixer DERR
C453 CAS TEST EOF
C45A accepter signe comme nombre entier
C45F CAS IN CHAR
C462 pas d'erreur ?
C468 sortir message d'erreur
C46B 'erreur disquette'
****** fixer POS sur un
C472 KM READ CHAR
****** tester si interruption avec 'ESC'

C475 KM READ CHAR
C479 'Break'
C47C attendre deuxième frappe de touche
C47F 'ESC', alors interruption
C485 adresse de la routine Break Event
C488 BASIC-ROM-Select
C48E KM ARM BREAK
****** Break-Event-Routine
C495 KM READ CHAR
C498 aucune touche appuyée ?
C49A Break par 'ESC'
C49C ignorer touches frappées avant 'ESC'
C49E attendre deuxième 'ESC'
****** attendre touche frappée après 'ESC'
****** attendre touche frappée après 'ESC'

C4A7 SOUND HOLD
C4AF TXT CUR ON
C4B2 KM WAIT CHAR
C4B5 'ESC'
C4BB TXT CUR OFF
C4C3 ' '
C4C5 KM CHAR RETURN
C4CA SOUND CONTINUE
C4DC KM DISARM BREAK
****** instruction BASIC ORIGIN
C4E1 aller chercher 2 arguments
C4E6 suit virgule ?
C4E9 non
C4EB aller chercher 2 arguments
C4F0 tester si ','
C4F3 aller chercher 2 arguments
C4F8 GRA WIN HEIGHT
C4FE GRA WIN WIDTH
C504 GRA SET ORIGIN
C509 fin de l'instruction ?
C510 GRA CLEAR WINDOW
******* instruction BASIC FILL
C515 aller chercher argument < 16

C51A Garbage Collection
C51D calculer place mémoire libre
C520 au moins 29 octets
C523 comparaison HL <> BC
C526 sinon 'Memory full'
C528 sortir message d'erreur
C52D FILL
****** instruction BASIC MOVE
C532 GRA MOVE ABSOLUTE
****** instruction BASIC MOVER
C537 GRA MOVE RELATIVE
****** instruction BASIC DRAW
C53C GRA LINE ABSOLUTE
****** instruction BASIC DRAWR
C541 GRA LINE RELATIVE
****** instruction BASIC PLOT
C546 GRA PLOT ABSOLUTE
****** instruction BASIC PLOTR
C54B GRA PLOT ABSOLUTE
C54F aller chercher 2 arguments entiers
C552 suit virgule ?
C555 non
C557 ','
C55C suit virgule ?
C55F non
C563 aller chercher valeur 8 bits < 4
C567 SCR ACCESS
C56F saut dans (BC)
****** fonction BASIC TEST
C574 GRA TEST ABSOLUTE
****** fonction BASIC TESTR
C579 GRA TEST RELATIVE
C57D aller chercher 2 arguments
C580 tester si ')'
C587 saut dans (BC)
C58A accepter contenu accu comme nombre entier
****** aller chercher 2 arguments entiers
C58F aller chercher valeur 16 bits -32768 à 32767
C593 tester si ','

C596 aller chercher valeur 16 bits -32768 à 32767
C59A résultat dans BC
****** instruction BASIC GRAPHICS
C59D 'PAPER'
C5A1 tester si encore un caractère
C5A4 'PEN'
C5A5 ','
C5AA suit virgule ?
C5AE aller chercher valeur 8 bits < 2
****** GRAPHICS PAPER
C5B4 ignorer les espaces
C5B7 aller chercher argument < 16
C5BA GRA SET PAPER
****** GRAPHICS PEN
C5BD aller chercher argument < 16
C5C0 GRA SET PEN
****** instruction BASIC MASK
C5C3 ','
C5C7 aller chercher argument 8 bits
C5CD suit virgule ?
C5D1 aller chercher valeur 8 bits < 2
****** instruction BASIC FOR
C5D7 lire variable
C5DD chercher NEXT correspondant
C5E0 ranger adresse
C5E6 chercher boucle FOR-NEXT ouverte
C5E9 trouvé, fixer pointeur de pile BASIC
C5ED fin de l'instruction ?
C5F0 défaut zéro
C5F3 non, aller chercher variable
C5FC comparaison HL<>DE
C5FF 'Unexpected NEXT'
C603 adresse de ligne actuelle dans HL
C607 fixer adresse de ligne actuelle
C610 22 octets, type 5 'Real'
C616 16 octets, type 2 'Integer'
C61A 'type mismatch'
C61C sortir message d'erreur
C61F nombre octets dans A

C620 reserver place dans pile BASIC
C624 adresse de variable sur pile BASIC
C628 tester si '='
C62B aller chercher expression

C62F comparer type de variable
C633 mémoire provisoire pour variable FOR
C636 copier variable dans HL
C63A tester si encore un caractère
C63D 'TO'
C63E aller chercher expression
C643 comparer type de variable
C646 valeur finale sur pile BASIC
C64C un comme valeur STEP défaut
C64F accepter nombre entier HL
C653 prochain caractère
C654 'STEP' ?
C656 non
C658 ignorer les espaces
C65B aller chercher expression
C65F comparer type de variable
C663 copier variable dans (HL)
C666 aller chercher signe
C66A signe de valeur STEP sur pile BASIC
C66E fin de l'instruction, sinon 'Syntax error'
C673 adresse de l'instruction FOR sur pile BASIC
C677 adresse de ligne actuelle dans HL
C67C adresse de ligne de FOR sur pile BASIC
C681 adresse de l'instruction NEXT sur pile BASIC
C689 adresse de ligne instruction NEXT sur pile BASIC
C68C #10 ou #16 pour Integer/Real sur pile
C68E pointeur sur mémoire provisoire
C691 ramener variable FOR
C695 flag pour premier parcours
C699 fixer adresse de ligne actuelle
C69F à l'instruction NEXT
C6A1 sortir message d'erreur
C6A4 'Unexpected NEXT'
******* instruction BASIC NEXT

C6A5 additionner
C6A7 flag pour incrément
C6AB chercher boucle FOR-NEXT ouverte
C6B1 fixer pointeur de pile BASIC
C6B6 tester si fin de boucle
C6BE pointeur de programme dans DE
C6C2 adresse de ligne dans HL
C6C5 fixer adresse de ligne actuelle
C6CA pointeur de pile BASIC
C6CD plus 5
C6CF pointeur de programme dans 'NEXT'
C6D2 fixer pointeur de pile BASIC
C6D6 suit virgule ?
C6D9 oui, prochaine boucle NEXT
****** chercher boucle FOR-NEXT ouverte
C6DC pointeur de pile BASIC
C6EB 'WHILE-WEND' ?
C6F8 comparaison HL <> DE
C70A Integer ?
C70C oui
C713 fixer type et adresse de variable
C717 flag pour premier parcours
C71B oui, sauter addition
C71F addition
C729 copier variable dans (HL)
C730 comparaison arithmétique
C734 10
C73E premier parcours ?
C742 oui, sauter addition
C749 aller chercher valeur STEP dans HL
C74C Integer-Addition HL := HL + DE
C74F 'Overflow'
C751 sortir message d'erreur

C761 comparaison entiers
****** instruction BASIC IF
C76A aller chercher expression
C76D 'GOTO'
C771 tester si encore un caractère

C774 'THEN'
C778 chercher fin de la ligne ou branche ELSE
C77C fin de l'instruction ?
C77F oui
C780 numéro de ligne
C782 oui, à l'instruction GOTO
C784 adresse de ligne ?
C786 non, exécuter instruction BASIC
****** instruction BASIC GOTO
C789 aller chercher adresse de ligne
C78D accepter adresse comme pointeur de programme
****** instruction BASIC GOSUB
C78F aller chercher adresse de ligne
C794 marque pour GOSUB normal
C796 ranger adresse du sous-programme
C797 6 octets
C799 reserver place dans pile BASIC
C79F adresse de l'instruction dans 'GOSUB'
C7A0 sur pile BASIC
C7A3 adresse de ligne actuelle dans HL
C7A8 adresse de ligne sur pile BASIC
C7AB marque pour 'GOSUB'
C7B1 pointeur de programme sur sous-programme
******* instruction BASIC RETURN
C7B4 chercher 'GOSUB' sur pile BASIC
C7B7 restaurer pointeur de pile BASIC
C7BA octet type
C7BD adresse de l'instruction dans 'GOSUB'
C7BE aller chercher dans DE
C7C1 adresse de ligne dans HL
C7C4 fixer actuel numéro de ligne
C7C8 octet type
C7C9 plus petit que un ?
C7CB oui, GOSUB normal
C7CC un, alors GOSUB dans AFTER/EVERY
C7CF à la routine event
C7D6 aller chercher marque de pile BASIC
C7DB restaurer pointeur de pile BASIC
C7E0 'GOSUB'

C7E6   sortir message d'erreur
C7E9   'Unexpected RETURN'
****** instruction BASIC WHILE
C7EB   chercher WEND correspondant
C7EE   ranger adresse
C7F0   adresse de ligne pour 'WHILE-WEND'
C7F6   fixer pointeur de pile BASIC
C7F9   7 octets
C7FB   reserver place dans pile BASIC
C7FF   adresse de ligne actuelle dans HL
C804   adresse de ligne sur pile BASIC
C809   adresse dans 'WEND' sur pile BASIC
C810   adresse de la condition WHILE
C811   sur pile BASIC
C813   marque pour 'WHILE'
C816   fixer pointeur de pile BASIC
C81B   tester condition WHILE
****** instruction BASIC WEND
C81D
C822   'Unexpected WEND'
C824   sortir message d'erreur
C82C   fixer pointeur de pile BASIC
C82F   adresse de ligne actuelle dans HL
C832   adresse de ligne pour WHILE-WEND
C83B   fixer adresse de ligne actuelle
C848   aller chercher expression
C84B   aller chercher signe
C84F   condition remplie ?
C850   adresse de ligne pour WHILE-WEND
C853   fixer comme adresse de ligne actuelle
C858   libérer place dans pile BASIC
****
C860   pointeur de pile BASIC
C87B   comparaison HL <> DE
******* instruction BASIC ON
C885   'ERROR'
C88A   aller chercher valeur 8 bits
C88F   'GOTO'
C894   tester si encore un caractère

C897 'GOSUB'

C899 ignorer les espaces suivants
C89C décrémenter compteur
C89F aller chercher numéro de ligne dans DE
C8A2 suit virgule ?
****** traitement event (AFTER/EVERY)
C8B5
C8B9 KL NEXT SYNC
C8BC aucun évènement en attente ?
C8BE ranger priorité
C8C2 annuler bit 7
C8C8 adresse du bloc event
C8C9 KL DO SYNC
C8D4 KL DONE SYNC
C8D9 prochain Event
C8E0 autoriser interruption par 'ESC'
C8ED 'Break'
C8F2 à la boucle de l'interpréteur
C8FC adresse ON-BREAK
C901 numéro de ligne dans HL
C906 mode direct ?
C909 SOUND CONTINUE
C915 tester si encore un caractère
C918 'GOSUB'
C919 aller chercher adresse de ligne
C91D dans BC
C920 10
****** Event-Routine
C929
C92D aller chercher numéro de ligne/mode direct ?
C932 oui
C934 octet type pour AFTER/EVERY-GOSUB
C937 instruction GOSUB
C93A adresse de l'instruction actuelle
C949 adresse de l'instruction actuelle
C95A -8
C95E KL DONE SYNC
C968 -4

C96C KL DONE SYNC
C96F autoriser interruption par 'Break'
C976 à la boucle de l'interpréteur
****** instruction BASIC ON BREAK
C979
C97C ignorer les espaces
C97F 'CONT'
C984 'STOP'
C986 valeur défaut zéro pour Stop
C98B tester si encore un caractère
C98E 'GOSUB'
C98F aller chercher adresse de ligne
C993 adresse ON-BREAK
C997 KM DISARM BREAK
******* instruction BASIC DI
C99A
C99B KL EVENT DISABLE
******* instruction BASIC EI
C9A0
C9A1 KL EVENT ENABLE
****** reset SOUND et event
C9A6 SOUND RESET
C9A9 adresse de base du bloc event
C9AC 4 Timer
C9AF KL DEL TICKER
C9B3 18
C9B6 additionner
C9B7 prochain timer
C9B9 KM DISARM BREAK
C9BC KL SYNC RESET
C9C2 annuler adresse ON-BREAK
C9C5 autoriser interruption par BREAK
C9C8 adresse de la Sound-Queue
C9D4 adresse du bloc event
C9DF BASIC-ROM-Select
C9E1 ADresse de la Event-Routine
C9E4 KL INIT EVENT
****** instruction BASIC ON SQ
C9F8 tester si '('

C9FB aller chercher valeur 8 bits
C9FF calculer adresse de la Sound-Queue
CA05 tester si ')'
CA08 aller chercher 'GOSUB' et adresse
CA0E SOUND ARM EVENT

****** calculer adresse de la Sound-Queue
CA13 bit 0 mis ?
CA18 bit 1 mis ?
CA1D bit 2 mis ?
CA22 'Improper argument'
****** instruction BASIC AFTER
CA25 aller chercher valeur 16 bits 0 - 32767
CA28 Recharge Count sur zéro
****** instruction BASIC EVERY
CA2D aller chercher valeur 16 bits 0 - 32767
CA30 comme Count et
CA31 Recharge Count
CA34 suit virgule ?
CA37 valeur défaut zéro
CA3A oui, aller chercher valeur entière avec signe
CA3E aller chercher dans Timer# adresse du bloc event
CA42 additionner 6 octets pour Tickerblock
CA47 aller chercher 'GOSUB' et adresse
CA4E KL ADD TICKER
****** fonction BASIC REMAIN
CA53 CINT
CA56 aller chercher adresse du bloc event
CA59 KL DEL TICKER
CA5C trouvé ?
CA5E non, zéro
CA62 accepter nombre entier dans HL
****** calculer adresse du bloc event
CA65
CA66 octet fort égale zéro ?
CA67 non, 'Improper argument'
CA6A supérieur ou égal 4 ?
CA6C oui, 'Improper argument'
CA70 * 18

CA74 adresse de base table event
CA77 plus Offset
****** chercher NEXT correspondant
CA79
CA7A adresse de ligne actuelle dans HL
CA7F compteur pour imbrication
CA81 numéro d'erreur pour 'NEXT missing'
CA87 ignorer les espaces
CA8A 'NEXT'
CA8F 'FOR'
CA93 incrémenter imbrication
CA94 chercher encore
CA99 adresse de ligne actuelle dans HL
CA9D fixer adresse de ligne actuelle
CAA1 décrémenter imbrication
CAA2 NEXT correspondant trouvé ?
CAA4 ignorer les espaces
CAA7 fin de ligne ?
CAAB chercher variable
CAB0 suit virgule ?
CAB3 non
CAB5 sinon prochaine variable dans NEXT
CAB8 trouvé NEXT correspondant
CABA oui
CABD adresse de ligne actuelle dans HL
CAC1 fixer adresse de ligne actuelle
CAC6 chercher encore
CAC9 ignorer les espaces
****** chercher WEND correspondant
CACC
CACE adresse de ligne actuelle dans HL
CAD2 compteur pour imbrication
CAD4 incrémenter
CAD5 numéro d'erreur pour 'WEND missing'
CADB ignorer les espaces
CADF 'WHILE'
CAE1 incrémenter imbrication
CAE3 'WEND'
CAE7 décrémenter imbrication

CAE9   ignorer les espaces
CAEC   ignorer les espaces
CAEF   aller chercher ligne d'entrée
CAF3   sélectionné stream 0
CAF6   initialiser pointeur de pile
CAF9   à la boucle de l'interpréteur
****** aller chercher ligne d'entrée
CAFC   pointeur sur buffer d'entrée
CAFF   annuler contenu buffer
CB01   aller chercher ligne d'entrée
****** editer ligne
CB04   pointeur sur buffer d'entrée
CB07   editer ligne
CB0A   sortir LF
****** aller chercher ligne d'entrée sur la disquette
CB0D
CB0E   pointeur sur buffer d'entrée
CB18   DISK IN CHAR
CB1D   CR
CB25   LF
CB2D   sortir message d'erreur
CB30   'Line too long'
CB32   LF
****** annuler numéro d'erreur
CB3A
****** fixer numéro d'erreur
CB3B   erreur disquette
CB3E   numéro d'erreur
CB41   adresse de ligne actuelle dans HL
CB44   comme ERROR-Line
****** sortir message d'erreur
CB48   adresse de retour dans HL
CB49   aller chercher caractère dans instruction CALL
CB4A   comme numéro d'erreur, sortir message
****** sortir 'Syntax error'
CB4C   numéro d'erreur pour 'Syntax error'
CB4E   sortir message d'erreur
****** sortir 'Improper argument'
CB50   numéro d'erreur pour 'Improper argument'

CB52 sortir message d'erreur
****** instruction BASIC ERROR
CB54 aller chercher valeur 8 bits différ. de 0
CB58 fixer numéro et ligne d'erreur
CB5B adresse de l'instruction actuelle
CB5E pointeur de programme dans ERROR
CB61 ranger adresse de ligne et pointeur de programme
CB67 pointeur de pile sur C000
CB70 initialiser descripteur de pile
CB79 adresse de la routine ON-ERROR
CB7D flag pour en traitement d'erreur
CB8B à la boucle de l'interpréteur

CB90 numéro d'erreur
CB93 calculer adresse des messages d'erreur
CB99 comme actuel numéro de ligne
CBA3 erreur disquette
CBAA au mode READY
CBAD adresse de la ligne ERROR
CBB0 aller chercher numéro de ligne dans HL
CBBA pointeur sur 'Division by zero'
CBBD numéro d'erreur
CBC3 pointeur sur 'Overflow'
CBC6 numéro d'erreur
CBCA adresse de la routine ON-ERROR
CBD0 numéro d'erreur dans accu
CBD1 sortir message d'erreur
CBD4 numéro stream sur zéro
CBD5 stream sélectionné
CBD8 ranger ancien numéro stream
CBDA sortir message d'erreur
CBDE sortir LF
CBE1 ancien numéro stream
CBE2 sélectionné
CBE9 initialiser écran
CBEC sortir 'Undefind line'
CBEF sortir numéro de ligne
CBF2 sortir 'in numéro de ligne'
CBF4 'Undefined line ',0

CC04 pointeur sur 'Break'
CC0A initialiser écran
CC0D sortir message d'erreur
CC10 aller chercher adresse de ligne
CC13 mode direct ?
CC15 pointeur sur ' in '
CC18 sortir chaîne
CC1C sortir numéro de ligne
CC1F 'Break'
CC24 ' in ',0
****** instruction BASIC STOP
CC29
CC2B sortir 'Break in numéro de ligne'
CC32 au mode READY
****** instruction BASIC END
CC34
****** erreur disquette
CC3A ranger erreur disque
CC3D sortir message d'erreur
CC40 'Nr 32'
CC4A au mode READY
CC66 au mode READY
CC6A aller chercher numéro de ligne dans HL
CC6E mode direct ?
CC70 fin de l'instruction ?
CC7B fixer adresse de ligne actuelle
CC87 mode direct ?
CC8A oui
CC8B adresse de ligne dans HL
CC8E adresse de ligne après interruption
CC92 pointeur de programme après interruption
****** instruction BASIC CONT
CC96
CC97 pointeur de programme après interruption
CC9B tester si mode direct
CC9C 'Cannot CONTinue'
CC9E sortir message d'erreur
CCA2 adresse de ligne après interruption
CCA5 fixer adresse de ligne actuelle

CCA8 SOUND CONTINUE
CCAC à la boucle de l'interpréteur
CCB0 annuler flag pour en traitement d'erreur
CCB6 adresse de la routine ON-ERROR
****** ON ERROR
CCBB ignorer les espaces
CCBE tester si encore un caractère
CCC1 'GOTO'
CCC2 aller chercher numéro de ligne dans DE
CCC6 chercher ligne BASIC DE
CCCB fixer adresse de la routine ON-ERROR
****** instruction BASIC ON ERROR GOTO 0
CCCD adresse ON-ERROR sur zéro
CCD0 en traitement d'erreur ?
CCD4 non
CCD5 sortie d'erreur
****** instruction BASIC RESUME
CCD8
CCDA 'NEXT'
CCDE aller chercher adresse de ligne
CCE2 en traitement d'erreur ?
CCE8 à la boucle de l'interpréteur
CCEB en traitement d'erreur ?
CCEF à la boucle de l'interpréteur
CCF2 ignorer les espaces
CCF6 en traitement d'erreur
CCFA ignorer reste de la ligne
CCFD en traitement d'erreur ?
CD01 'Unexpected RESUME'
CD03 non, sortir message d'erreur
CD07 annuler numéro ERROR
CD0A annuler flag pour en traitement d'erreur
CD0D adresse de la ligne ERROR
CD10 comme adresse de ligne actuelle
CD13 pointeur de programme après ERROR

****** messages d'erreur
****** sortir message d'erreur
CE76 adresse de base des messages d'erreur

CE79 fixer pointeur sur message d'erreur
CE7D aller chercher caractère des messages d'erreur
CE7E annuler bit 7
CE80 caractère imprimable ?
CE82 oui, sortir
CE85 non, fixer message d'erreur
CE89 aller chercher caractère encore une fois
CE8A incrémenter pointeur
CE8B tester bit 7
CE8C pas mis, sortir encore
****** fixer pointeur DE sur message d'erreur
CE8F
CE95 numéro zéro ?
CE96 terminé
CE98 numéro d'erreur dans B
CE99 aller chercher caractère
CE9A incrémenter pointeur
CE9B tester bit 7
CE9C pas mis, ignorer message
CE9E prochain message d'erreur
CEA0 DE pointe maintenant sur début du message
****** aller chercher valeur 8 bits
CEBB aller chercher valeur entière avec signe
CEBF octet fort
CEC1 differ. de zéro, 'Improper argument'
CEC4 accepter octet faible
******* aller chercher valeur 8 bits différente de zéro
CEC6 aller chercher valeur entière avec signe
CECC différ. de zéro ?
CECE 'Improper argument'
******* aller chercher valeur 16 bits 0 à 32767
CED1 aller chercher valeur entière avec signe
CED5 octet fort
CED6 tester bit 15
CED7 mis, 'Improper argument'
******* aller chercher valeur entière avec signe
CEDB aller chercher expression
CEE0 CINT
CEE6 aller chercher expression

CEE9 tester si chaîne
CEEC non
****** aller chercher valeur 16 bits, expression d'adresse
CEF8 aller chercher expression
CEFE UNT
****** aller chercher expression chaîne et paramètre
CF06 aller chercher expression
CF09 aller chercher param. de chaîne
****** aller chercher expression chaîne
CF0C aller chercher expression
CF0F type chaîne, sinon 'Type mismatch'
****** aller chercher numéro de zone de ligne
CF12 1 et
CF15 65535 comme défaut
CF18 suit virgule ?
CF1B non, fin de l'instruction ?
CF1E oui
CF1F '#'
CF22 '-'
CF26 aller chercher numéro de ligne dans DE

CF2A et dans BC
CF2C suit virgule ?
CF2F oui
CF30 tester si encore un caractère
CF33 '-'
CF34 65535 comme valeur finale défaut
CF38 suit virgule ?
CF3B oui
CF3C aller chercher numéro de ligne dans DE
CF3F suit virgule ?
CF46 'Improper argument'
******* aller chercher numéro de ligne dans DE
CF4B type de constante
CF4E valeur dans DE
CF50 numéro de ligne ?
CF52 oui, terminé
CF54 adresse de ligne ?
CF56 non, 'Syntax error'

CF5A HL pointe sur début de ligne
CF5F numéro de ligne dans DE
CF62 ignorer les espaces
****** aller chercher expression
CF65
CF66 code de hiérarchie zéro
CF68 aller chercher terme
CF6D ignorer les espaces
****** aller chercher terme
CF70
CF72 aller chercher expression
CF78 opérateur
CF79 '>'
CF7B plus petit ?
CF7C 'NOT'
CF7E supérieur ou égal ?
CF7F '+'
CF81 plus petit, alors opérateur de comparaison
CF83 '+', alors tester si chaîne
CF86 pas chaîne
CF8A descripteur de chaîne
CF8D sur pile
CF8E aller chercher expression
CF91 type chaîne, sinon 'Type mismatch'
CF95 addition de chaîne
CF98 traiter prochain terme
****** opérateurs arithmétiques
CF9A
CF9B moins #F4
CF9E fois 4
CFA2 plus #CFF0, adresse table
CFA9 code de hiérarchie
CFAB plus petit, terminé
CFAD placer résultat sur pile
CFB3 code de hiérarchie
CFB4 aller chercher terme
CFC0 reserver place dans pile BASIC
CFC3 JP (DE), exécuter opération
CFC6 traiter prochain terme

****** opérateurs de comparaison
CFC8
CFCD Token
CFCE moins Offset
CFD1 tester si chaîne
CFD4 adresse pour comparaisons arithmétiques
CFD7 pas chaîne
CFDA descripteur de chaîne
CFDD sur pile
CFDF code de hiérarchie
CFE1 aller chercher terme
CFE7 comparaison de chaîne
CFEB aller chercher résultat des comparaisons
CFEE traiter prochain terme
****** opérateurs BASIC codes de hiérarchie + adresses
CFF0 F4, '+'
CFF3 F5, '-'
CFF6 F6, '*'
CFF9 F7, '/'
CFFC F8, '^'
CFFF F9, 'Backslash'
D002 FA, 'AND'
D005 FB, 'MOD'
D008 FC, 'OR'
D00B FD, 'XOR'
****** comparaison arithmétique
D00E
D012 comparaison arithmétique
D01D accepter signe
****** '-' signe négatif
D020 code de hiérarchie
D022 aller chercher terme
D026 changer signe
****** opérateur BASIC NOT
D02B code de hiérarchie
D02D aller chercher terme
D031 opérateur NOT
****** aller chercher expression
D036 ignorer les espaces

****** aller chercher expression
D039 'Operand missing'
D03D aller chercher variable
D041 aller chercher valeur numérique
D043 '"'
D045 aller chercher chaîne
D048 fonction ?
D04A au calcul de fonction
D04E adresse de base de la table
D051 rechercher dans la table
D055 ignorer les espaces, saut à fonction
D058 sortir message d'erreur
D05B 'Operand missing'
****** fonctions spéciales
D05C nombre des entrées de la table
D05D pas trouvé, 'Syntax error'
D05E '-'
D062 '+'
D065 '('
D068 'NOT'
D06B 'ERL'
D06E 'FN'
D071 'MID$'
D074 '@'

****** aller chercher variable
D077 aller chercher adresse de variable
D07A pas encore définie ?
D07C type de variable
D07E chaîne ?
D087 chaîne ?
D089 annuler variable
D08C pointeur sur zéro
D08F comme descripteur de chaîne
D094 longueur de chaîne zéro
******* aller chercher valeur numérique
D095 ôter Offset
D09A plus petit que 10 ?
D09C oui, aller chercher chiffre

D0A0 valeur un octet ?
D0A2 oui
D0A6 valeur deux octets (déc, hex, bin) ?
D0A8 oui
D0AA valeur à virgule flottante ?
D0AC oui
D0AE 'Syntax error'
D0B1 'Real'
D0B3 fixer type de variable
****** aller chercher valeur deux octets
D0B9
****** aller chercher valeur à virgule flottante
D0C0
D0CD type de variable sur 'Real'
D0D1 ignorer les espaces
******* '(' aller chercher terme entre parenthèses
D0D4 aller chercher expression
D0D7 tester si ')'
****
D0DA 'Syntax error'
******* calcul de fonction
D0DD incrémenter pointeur de programme
D0DE aller chercher Token
D0DF ignorer les espaces
D0E2 tester Token
D0E5 40 - 49, variable réservée
D0E9 aller chercher adresse de variable réservée
D0EC tester si '('
D0F0 Token fois 2
D0F6 'Syntax error'
D0FA calculer fonction
D0FC aller chercher argument de fonction en parenthèses
D100 calculer fonction
****** calculer fonction
D105 adresse des fonctions
D10A annuler octet fort
D10C additionner Token fois 2
D111 exécuter fonction
****** aller chercher adresse de variable réservée

D113 doubler Token
D115 adresse de base d'Offset de table
****** adresses des variables réservées
D11A 40, EOF
D11C 41, ERR
D11E 42, HIMEM
D120 43, INKEY$
D122 44, PI
D124 45, RND
D126 46, TIME
D128 47, XPOS
D12A 48, YPOS
D12C 49, DERR
****** variable réservée DERR
D12E numéro erreur disque
****** variable réservée ERR
D133 numéro ERROR
D137 accepter contenu accu comme nombre entier
****** variable réservée TIME
D13D KL TIME PLEASE
D140 convertir valeur 4 octets en format virgule flottante
****** variable réservée ERL
D146 aller chercher numéro de ligne ERROR
****** variable réservée HIMEM
D14B
D14C HIMEM
D14F accepter valeur
****** '@', pointeur de variable
D151 aller chercher adresse de variable
D154 pas défini, 'Improper argument'
D15A chaîne ?
D15F accepter valeur
****** variable réservée XPOS
D164
D165 GRA ASK CURSOR
D168 valeur de colonne dans HL
****** variable réservée YPOS
D16C GRA ASK CURSOR
D16F accepter nombre entier dans HL

******* instruction BASIC DEF
D174 tester si encore un caractère
D177 'FN'
D179 aller chercher numéro de ligne dans HL
D17D 'Invalid direct command'
D17F sortir message d'erreur
D182 chercher fonction
D18A ignorer reste de l'instruction
****** fonction BASIC FN
D18D chercher fonction
D199 'Unknown user function'
D19B sortir message d'erreur
D1A2 '('
D1A6 ignorer les espaces
D1AA tester si '('
D1B3 aller chercher expression
D1B8 affecter valeur à une variable
D1BC suit virgule ?
D1BF non
D1C2 tester si ','
D1C5 prochaine variable
D1C7 tester si ')'
D1CB tester si ')'
D1D1 tester si '='
D1D4 aller chercher expression
D1D7 'Syntax error'
D1DA tester si chaîne

D1E5 tester si même type de variable
****** fonctions BASIC avec plusieurs arguments
D1E8 71, BIN$
D1EA 72, DEC$
D1EC 73, HEX$
D1EE 74, INSTR
D1F0 75, LEFT$
D1F2 76, MAX
D1F4 77, MIN
D1F6 78, POS
D1F8 79, RIGHT$

D1FA 7A, ROUND
D1FC 7B, STRING$
D1FE 7C, TEST
D200 7D, TESTR
D202 7E, COPYCHR$
D204 7F, VPOS
****** adresses des fonctions BASIC
D206 00, ABS
D208 01, ASC
D20A 02, ATN
D20C 03, CHR$
D20E 04, CINT
D210 05, COS
D212 06, CREAL
D214 07, EXP
D216 08, FIX
D218 09, FRE
D21A 0A, INKEY
D21C 0B, INP
D21E 0C, INT
D220 0D, JOY
D222 0E, LEN
D224 0F, LOG
D226 10, LOG10
D228 11, LOWER$
D22A 12, PEEK
D22C 13, REMAIN
D22E 14, SGN
D230 15, SIN
D232 16, SPACE$
D234 17, SQ
D236 18, SQR
D238 19, STR$
D23A 1A, TAN
D23C 1B, UNT
D23E 1C, UPPER$
D240 1D, VAL
****** fonction BASIC MIN
D242 flag pour MIN

****** fonction BASIC MAX
D246 flag pour MAX
D248 aller chercher expression
D24B suit virgule ?
D24E non, tester si ')', terminé
D251 placer variable sur pile BASIC
D254 aller chercher expression
D259 libérer place dans pile BASIC
D25E comparaison arithmétique
D267 aller chercher résultat des comparaisons
D26B prochain argument
****** fonction BASIC ROUND
D26D aller chercher expression
D270 et placer sur pile BASIC
D273 suit virgule ?
D276 défaut zéro
D279 oui, aller chercher valeur entière avec signe
D27C tester si ')'
D281 39
D284 additionner
D285 79
D288 comparaison HL <> DE
D28B supérieur, 'Improper argument'
D290 libérer place dans pile BASIC
D293 nombre de chiffres d'arrondissage dans B
D294 arrondir nombre
****** instruction BASIC CAT
D29B interrompre I/O disque
D2A1 DISK CATALOG
D2A4 ranger erreur disquette
****** instruction BASIC OPENOUT
D2AB
D2B4 DISK OUT OPEN
****** instruction BASIC OPENIN
D2B7
D2BD sortir message d'erreur
D2C0 'File type error'
D2C1 aller chercher nom de fichier
D2C7 DISK IN OPEN

D2CD aller chercher expression et param. chaîne
D2D2 tester si messages système
D2D5 ranger erreur disquette
D2DA sortir message d'erreur
D2DD 'File already open'
****** tester si messages système
D2DE
D2E0 pas nom de fichier ?
D2E3 premier caractère du nom
D2E4 '!'
D2E8 non
D2EA fixer pointeur sur second caractère
D2EB décrémenter longueur
D2EC inverser flag
D2ED CAS NOISY
****** instruction BASIC CLOSEIN
D2F0
D2F1 DISK IN CLOSE
****** instruction BASIC CLOSEOUT
D2F8
D2F9 DISK OUT CLOSE
D2FC ranger erreur disquette
****** interrompre I/O disque
D303
D306 DISK IN ABANDOM
D30C DISK OUT ABANDOM
******* instruction BASIC SOUND
D316 aller chercher valeur 8 bits
D319 état canal
D31C tester si ','
D31F aller chercher argument 0 à 4095
D322 période de note
D326 suit virgule ?
D329 valeur défaut 20
D32C oui, aller chercher valeur entière avec signe
D32F durée
D333 max. 15, défaut 12
D336 aller chercher arguments s'il y en a
D339 volume

D33C max. 15, défaut 0
D33E aller chercher arguments s'il y en a
D341 courbe d'enveloppe de volume
D344 aller chercher arguments s'il y en a
D347 courbe d'enveloppe de note
D34A max. 31, défaut 0
D34C aller chercher arguments s'il y en a
D34F période de bruit
D352 fin de l'instruction, sinon 'Syntax error'
D356 adresse du bloc de paramètres SOUND
D359 SOUND QUEUE
D35F à la boucle de l'interpréteur
****** aller chercher s'il y a lieu valeur 8 bits
D362 suit virgule ?
D365 charger valeur défaut
D366 pas virgule, terminé

D368 ','
D36C aller chercher valeur 8 bits
D36F comparer avec valeur maximum
D370 plus petit, ok
D371 'Improper argument'
****** instruction BASIC RELEASE
D373 8
D375 aller chercher valeur 8 bits < 8
D379 SOUND RELEASE
****** fonction BASIC SQ
D37E CINT
D383 tester bit 0
D384 mis ?
D386 tester bit 1
D387 mis ?
D389 tester bit 2
D38A pas mis, 'Improper argument'
D38C octet fort supérieur zéro ?
D38D oui, 'Improper argument'
D390 SOUND CHECK
D393 accepter contenu accu comme nombre entier
****** aller chercher argument -128 à +127

D396 aller chercher valeur entière avec signe
D39E 'Improper argument'
******* instruction BASIC ENV
D3A1 aller chercher valeur 8 bits différente de zéro
D3A4 supérieur ou égal 16 ?
D3A6 oui, 'Improper argument'
D3AC aller chercher paramètre
D3B1 adresse du bloc de paramètre
D3B5 SOUND AMPL ENVELOPE
D3BB '='
D3BF ignorer les espaces
D3C2 16
D3C4 aller chercher valeur 8 bits < 16
D3C7 fixer bit 7
D3CA tester si ','
D3CD aller chercher valeur 16 bits
D3D0 128
D3D2 aller chercher valeur 8 bits < 128
D3D5 aller chercher 2 arguments
****** instruction BASIC ENT
D3D7 aller chercher argument -128 à +127
D3DD zéro ?
D3E2 zéro ?
D3E3 'Improper argument'
D3E5 supérieur ou égal 16 ?
D3E7 'Improper argument'
D3ED aller chercher paramètre
D3F2 adresse du bloc de paramètre
D3FB SOUND TONE ENVELOPE
D401 '='
D405 ignorer les espaces
D408 aller chercher argument 0 à 4095
D412 240
D414 aller chercher valeur 8 bits < 240
D418 tester si ','
D41B aller chercher argument -128 à +127
D41F tester si ','
D422 aller chercher valeur 8 bits
****** aller chercher paramètre pour ENT & ENV

D428
D42B suit virgule ?
D432 JP (DE)
D439 adresse du bloc de paramètre
D44C fin de l'instruction, sinon 'Syntax error'
****** aller chercher argument 0 à 4095
D44F aller chercher valeur entière avec signe
D452 octet fort
D453 bit 12-15 mis ?
D455 oui, 'Improper argument'
****** fonction BASIC INKEY
D459 CINT
D45C 80
D45F comparaison HL <> DE
D462 'Improper argument'
D465 KM TEST KEY
D468 -1 si pas appuyée
D46D résultat dans L
D470 accepter nombre entier dans HL
****** fonction BASIC JOY
D473 KM GET JOYSTICK
D477 CINT
D483 accepter contenu accu comme nombre entier
D486 'Improper argument'
****** instruction BASIC KEY
D489 'DEF'
D48D aller chercher valeur 8 bits
D491 tester si ','
D494 aller chercher expression chaîne et paramètre
D497 longueur de chaîne dans C
D499 numéro de touche dans B
D49B adresse chaîne dans HL
D49C KM SET EXPAND
D4A0 'Improper argument'
****** KEY DEF
D4A3 ignorer les espaces
D4A6 80 comme valeur maximum
D4A8 aller chercher valeur 8 bits < 80
D4AC tester si ','

D4AF 2
D4B1 aller chercher argument < 2
D4BA KM SET REPEAT
D4BF KM SET TRANSLATE
D4C2 tester si encore un argument
D4C5 KM SET SHIFT
D4C8 tester si encore un argument
D4CB KM SET CONTROL
D4CE suit virgule ?
D4D1 non, terminé

D4D3 aller chercher valeur 8 bits
D4D9 saut en (HL)
****** instruction BASIC SPEED
D4DE 'WRITE'
D4E2 'KEY'
D4E4 KM SET DELAY
D4E9 'INK'
D4EB SCR SET FLASHING
D4EE 'Syntax error'
******* SPEED KEY & INK
D4F1
D4F2 ignorer les espaces
D4F5 aller chercher valeur 8 bits différente de zéro
D4F9 tester si ','
D4FC aller chercher valeur 8 bits différente de zéro
D503 saut en (BC)
****** SPEED WRITE
D508 ignorer les espaces
D50B 2
D50D aller chercher argument < 2
D511 167
D517 zéro ?
D519 non, doubler constante de durée
D51B CAS SET SPEED
******* variable réservée PI
D520
D521 fixer type sur 'Real'
D524 type de variable dans C, HL sur variable

D527 aller chercher PI
****** instruction BASIC DEG
D52C flag pour DEG
****** instruction BASIC RAD
D530 flag pour RAD
D531 fixer mode DEG/RAD
****** fonction BASIC SQR
D534 fonction SQR
****** opérateur BASIC '^'
D539
D53B CREAL
D53F mémoire provisoire pour variable à virgule flottante
D542 copier variable de (DE) dans (HL)
D548 fixer type et adresse de variable
D54C élévation à la puissance
D54F exécuter fonction
D552 pas d'erreur ?
D553 'Division by zero'
D556 'Overflow'
D559 'Improper argument'
****** exécuter fonction à virgule flottante
D55C
D55E CREAL
D562 exécuter fonction
****** fonction BASIC EXP
D563 fonction EXP
****** fonction BASIC LOG10
D568 fonction LOG10
****** fonction BASIC LOG
D56D fonction LOG
****** fonction BASIC SIN
D572 fonction SIN
****** fonction BASIC COS
D577 fonction COS
****** fonction BASIC TAN
D57C fonction TAN
****** fonction BASIC ATN
D581 fonction ATN
D586 'Random number seed ? ',0

******* instruction BASIC RANDOMIZE
D59C
D59E aller chercher expression
D5A5 'Random number seed ? '
D5A8 sortir
D5AB aller chercher ligne d'entrée
D5AE sortir LF
D5B1 lire entrée
D5B4 non valable, répéter
D5B6 ignorer espace, TAB et LF
D5BA non valable, répéter
D5BC CREAL
D5BF SET RANDOM SEED
****** variable réservée RND
D5C4
D5C5 '('
D5C9 ignorer les espaces
D5CC aller chercher expression
D5CF tester si ')'
D5D3 CREAL
D5D6 SGN
D5D9 differ. de zéro ?
D5DB aller chercher dernière valeur RND
D5E0 négatif, SET RANDOM SEED
D5E5 fixer type sur virgule flottante
D5E8 RND
****** restaurer pointeur de variable
D5ED annuler table
D5F0 fin du programme
D5F3 début des variables
D5F6 début des tableaux
D5F9 fin des tableaux
******* annuler table
D5FD base de la table
D600 54 = 2*27, A-Z plus fonctions
D602 annuler #ADB7 à #ADEC
D60A annuler #ADED à #ADF2
****** annuler flag pour FN
D611

****** calculer adresse table
D61A 'Z'+1
D61C début des variables
D620 moins 1
D621 fois 2
D624 plus #AD35
****** calculer adresse table pour tableau
D62A début des tableaux
D62E moins 1
D632 fois
D635 plus #ADED
****** toutes les variables sur type REAL
D63B 'AZ'
D63E type 'Real'
D641 nombre dans A
D642 plus petit 1, alors 'Syntax error'
D646 base de la table égale #ADB2
D64B lettre égale pointeur dans table
D64E toutes les lettres

****** instruction BASIC DEFSTR
D653 type 'chaîne'
****** instruction BASIC DEFINT
D657 type 'Integer'
****** instruction BASIC DEFREAL
D65B type 'Real'
D65D aller chercher lettre
D65E tester si lettre
D661 'Syntax error'
D663 dans BC (de - à)
D665 ignorer les espaces
D668 '-'
D66C ignorer les espaces
D66F tester si lettre
D672 'Syntax error'
D674 à
D675 ignorer les espaces
D678 fixer type de variable
D67B suit virgule ?

D67E oui, continuer
D681 'Syntax error'
D684 sortir message d'erreur
D687 'Subscript out of range'
D688 sortir message d'erreur
D68B 'Array already dimensioned'
D68C 2* #7C 'l'
D68E extension d'instruction
****** instruction BASIC LET
D691 aller chercher variable
D695 tester si '='
D698 aller chercher expression
D69D affecter valeur à une variable
****** affecter valeur à une variable
D6A2 type de variable
D6A3 et type du résultat
D6A6 comparer
D6A8 types correspondants, sinon 'Type mismatch'
D6AB tester si chaîne
D6AE non, copier variable dans (HL)
D6B2 gestion de chaîne
D6B6 accepter pointeur sur chaîne
****** instruction BASIC DIM
D6B9 dimensionnement
D6BC suit virgule ?
D6BF oui, prochaine variable
****** chercher variable
D6C2 lire nom de variable
D6C5 tester si variable dimensionnée
D6C8 aller chercher type de variable
****** aller chercher adresse de variable
D6CC lire nom de variable
D6CF tester si variable dimensionnée
D6D2 aller chercher type de variable
D6D5 première lettre
D6D6 calculer position table
****** chercher fonction
D6DE lire nom de variable
D6E4 calculer position table pour FN

D6EA créer fonction
D6EF lire nom de variable
D6F5 première lettre
D6F6 calculer position table
D6FC type de variable
D705 début des variables
D70C type de variable
D712 lire nom de variable
D715 tester si variable indexée
D718 aller chercher type de variable
D72A chercher tableau
D72E trouvé ?
D733 chercher tableau
D736 trouvé ?
****** chercher tableau
D75B type de variable
D77A fixer bit 6, 'FN'
D787 début des tableaux
D78B reserver place dans zone des variables
D78E incrémenter pointeur pour zone des tableaux

D7C6 LDIR
D7C9 type de variable
****** dimensionnement
D7E4 aller chercher nom de variable
D7E8 '('
D7EC ']'
D7EE 'Syntax error'
D7F6 type de variable
D7F9 calculer position table pour tableau
D7FC chercher tableau
D7FF trouvé, 'tableau already dimensioned'
****** tester si variable dimensionnée
D80A
D80C '('
D810 ']'
D817 début des variables
****** variable dimensionnée
D820
D827 début des tableaux
D830 type de variable
D833 calculer position table pour tableau
D836 chercher tableau
D839 pas trouvé ?
D845 'Subscript out of range'
D857 nombre des dimensions
D85C limite des tableaux dans DE
****** lire indices
D887
D888 ignorer les espaces
D88B type de variable
D88E ranger
D88F nombre des indices
D891 aller chercher valeur 16 bits 0 - 32767, index
D897 reserver place dans pile BASIC
D89B index sur pile BASIC
D89E incrémenter nombre des indices
D89F suit virgule ?
D8A2 oui, prochain index
D8A5 ')'

D8A9 '['
D8AB 'Syntax error'
D8AE ignorer les espaces
D8B2 restaurer type de variable
D8C4 fin des tableaux
D8C8 reserver place dans zone des variables
D8D5 type de variable
D8E3 10, valeur défaut pour index

D92B 2 octets
D92D libérer place dans pile BASIC
****** lire nom de variable
D935 déterminer type de variable
D93D variable déjà définie ?
D93E non
D941 ignorer lettres du nom
D942 tester bit 7
D943 dernière lettre ?
D945 ignorer les espaces suivants
D94B fixer pointeur sur type de variable
D988 type de variable
D991 #05 + #09 => #0D
D995 40
D997 reserver place dans pile BASIC
D99B 41
D99D déjà 40 caractères?
D99E oui, alors 'Syntax error'
D9A1 lire prochain caractère du nom
D9A3 convertir minuscules en majuscules
D9A7 dernier caractère ?
D9A8 non
D9AA fixer pointeur de pile BASIC
D9B0 ignorer les espaces suivants
****** déterminer type de variable
D9B3
D9B6 plus petit que #0B ?
D9B8 -#09, #0D => #05
D9BA '!', variable Real ?
D9BC fixer type sur 'Real'

D9BE 'Syntax error'
D9C0 '%', variable entière ?
D9C2 ou '$', chaîne ?
D9C4 non, 'Syntax error'
D9C7 'Real'
D9C9 ranger type de variable
****** actualiser table des tableaux
D9CD annuler table pour tableaux
D9D0 fin des tableaux
D9D4 début des tableaux
D9D7 comparaison HL <> DE
D9DA aucun tableau
D9E5 calculer position table pour tableau
******* instruction BASIC ERASE
D9F4
D9F7 annuler tableau
D9FA suit virgule ?
D9FD oui, prochain tableau
******* annuler tableau
DA00 lire nom de variable
DA04 type de variable
DA07 calculer position table pour tableau
DA0A chercher tableau
DA0D pas trouvé, 'Improper argument'
DA16 BC := HL - DE
DA1F actualiser table des tableaux

DA33 6 octets
DA35 reserver place dans pile BASIC
DA71 reserver place dans pile BASIC
DA75 déterminer type de variable
DA88 type de variable
DA8D reserver place dans pile BASIC
DAAC 26 lettres, 'A'
DAB0 première lettre du nom
DAB1 calculer position table
DAB8 prochaine lettre
DAB9 déjà toutes les lettres ?
DABD calculer position table pour tableau

DAC8 début des tableaux
DAE2 comparaison HL <> BC
DB10 saut en (HL)
****** instruction BASIC LINE
DB18 tester si encore un caractère
DB1B 'INPUT'
DB1C aller chercher numéro de canal
DB1F sortir évent. chaîne de dialogue
DB22 chercher variable
DB25 type 'chaîne', sinon 'Type mismatch'
DB2A aller chercher entrée sur périphérique actif
DB2D entrer chaîne dans descripteur de pile
DB31 affecter résultat à une variable
****** aller chercher entrée sur périphérique actif
DB36
DB39 aller chercher entrée sur la disquette
DB42 ';'
****** instruction BASIC INPUT
DB48 aller chercher numéro de canal
DB4B aller chercher entrée et convertir
DB4F chercher variable
DB59 suit virgule ?
DB5C oui, prochaine variable
****** aller chercher entrée et convertir
DB60
DB63 sortir évent. chaîne de dialogue

******
DB7E '?Redo from start', LF,0
****** sortir évent. chaîne de dialogue
DB90
DB91 ';'
DB93 ranger signe de séparation
DB96 ignorer les espaces
DB99 '"'
DB9B pas chaîne ?
DBA0 suit virgule ?
DBA3 oui
DBA4 tester si encore un caractère

DBA4 tester si encore un caractère
DBA7 ';'
DBAC '?'
DBAE sortir
DBB1 ' '
DBB3 sortir
DBC6 'type mismatch'
DBCE sortir message d'erreur
DBD1 'type mismatch'
DBD5 aller chercher nom et type de variable
DBDF 'chaîne'
DBE1 oui, aller chercher param. de chaîne
DBE5 suit virgule ?
DBEA non
DBEC ','
DBFD tester si chaîne
DC13 ignorer espace, TAB et LF
DC22 entrer chaîne dans Descriptor
DC25 ignorer espace, TAB et LF
DC28 '"'
DC2A lire chaîne
DC39 sortir message d'erreur
DC3C 'EOF met'
DC42 '"'
DC53 début du buffer d'entrée
DC56 premier caractère égale zéro
DC59 '"'
DC5F 'EOF met'
DC64 début du buffer d'entrée
DC6A JP (DE)
DC7F CR
DC82 '"'
DCA4 LF + CR
DCA7 CR ?
DCAA LF ?
DCBE ','
DCC1 CR
DCC4 ' '
DCC7 TAB

DCCA LF
****** instruction BASIC RESTORE
DCCD aucun numéro de ligne ?
DCCF aller chercher numéro de ligne dans DE
DCD3 chercher ligne BASIC DE
DCD7 fixer pointeur DATA
DCDA début du programme
DCDD comme pointeur DATA
****** instruction BASIC READ
DCDF
DCE0 pointeur DATA
DCE3 aller chercher prochain élément DATA

DCE7 chercher variable
DCEC ':'
DCF6 ','
DCFA adresse de ligne pendant instruction READ
DCFD fixer adresse de ligne actuelle
DD00 'Syntax error'
DD04 suit virgule ?
DD08 oui
DD0A pointeur DATA
DD10 ','
DD13 ignorer reste de la ligne
DD16 fin de ligne ?
DD17 non
DD1A longueur de ligne
DD1C zéro, fin du programme ?
DD1E 'DATA exhausted'
DD20 sortir message d'erreur
DD23 adresse de ligne pendant instruction READ
DD27 ignorer les espaces
DD2A 'DATA'
DD2C non, continuer à chercher
DD2F ranger signe
DD30 former valeur absolue
******* accepter signe B
DD3C
DD41 signe du résultat

DD42 négatif, alors changement de signe
DD47 inverser bit signe
******* addition entière HL := DE + HL
DD4F annuler flag carry
DD50 addition
DD53 résultat positif ?
DD54 fixer flags
******* soustraction entière HL := DE - HL
DD57 échanger opérandes
DD58 annuler flag carry
DD59 soustraction
DD5C résultat positif ?
DD5D fixer flags
******* multiplication entière avec signe
DD60 déterminer signe du résultat
DD63 multiplication non signée
DD66 accepter signe
******* déterminer signe du résultat
DD6C signe de HL
DD6D et signe de DE
DD6E dans B
DD70 former valeur absolue de DE
DD74 former valeur absolue de HL
******* multiplication entière sans signe
DD77
****** division entière avec signe
DDA1 Division HL := HL / DE
DDA4 accepter signe
******* calcul MOD integer
DDA8 ranger signe
DDA9 Division
DDAC reste dans HL
DDAD ramener signe
DDAE et accepter
****** Division HL := HL / DE, DE := reste
DDB0 déterminer signe du résultat
****** former valeur absolue
DDEF tester signe
DDF1 positif, déjà terminé

****** changement de signe integer
DDF2
******* SGN signe de HL
DDFE
****** comparaison HL <> DE
DE07 signe de HL
DE08 et signe de DE
DE0A comparer nombres avec même signe

******* tester si encore une virgule
DE1A ','
******* tester si parenthèse ouverte
DE1E '('
******* tester si parenthèse fermée
DE22 ')'
******* tester si signe égale
DE26 '='
******* tester si encore un caractère
DE2A aller chercher adresse de retour
DE2B aller chercher caractère
DE2D ramener pointeur de programme
DE2E comparer caractère
DE2F 'Syntax error'
****** ignorer les espaces
DE31
DE33 ' '
DE35 continuer à tester si espaces
DE37 fin de l'instruction ?
****** tester si fin de ligne, sinon 'Syntax error'
DE3C
DE40 'Syntax error'
****** tester si fin de l'instruction
DE42
****** tester si virgule
DE46
DE47 ignorer les espaces
DE4A ','
DE4D ignorer les espaces
****** ignorer espace, TAB et LF

DE52 aller chercher caractère
DE53 incrémenter pointeur
DE54 ' '
DE58 TAB
DE5C LF
DE60 décrémenter pointeur
****** boucle de l'interpréteur
DE62 adresse de l'instruction actuelle
DE65 fixer adresse de l'instruction actuelle
DE68 KL POLL SYNCHRONOUS
DE6B traitement event (AFTER/EVERY)
DE6E ignorer les espaces
DE71 exécuter instruction BASIC
DE74 lire texte programme
DE75 ':', fin de l'instruction ?
DE77 oui
DE79 'Syntax error'
DE7D longueur de ligne
DE7E égale zéro ?
DE80 oui, à l'instruction END
DE82 ranger adresse de ligne actuelle
DE86 flag TRACE mis ?
DE8A non
DE8C routine TRACE
DE8F au début de la boucle de l'interpréteur
DE91 à l'instruction END
****** exécuter instruction BASIC
DE94 Token fois 2
DE95 tester si extension d'instruction
DE9A token non valable, 'Syntax error'
DE9F plus #DEE5 (adresse table)
DEA7 adresse d'instruction sur pile
DEA9 ignorer les espaces, saut à instruction
DEAC 'Syntax error'
****** adresse de ligne actuelle sur zéro
DEAF zéro
DEB2 comme adresse de ligne actuelle
****** charger adresse de ligne actuelle
DEB6 adresse de ligne actuelle

****** test mode direct/aller chercher adresse de ligne
DEBA adresse de ligne actuelle
DEBF zéro, mode direct
DEC2 numéro de ligne dans HL
****** instruction BASIC TRON
DEC6 fixer flag
****** instruction BASIC TROFF
DECA annuler flag
****** routine TRACE
DECF ']'
DED1 sortir
DED5 adresse de ligne actuelle
DED9 numéro de ligne dans HL
DEDC sortir numéro de ligne
DEE0 '['
DEE2 sortir
******* adresses des instructions BASIC
DEE5 80, AFTER
DEE7 81, AUTO
DEE9 82, BORDER
DEEB 83, CALL
DEED 84, CAT
DEEF 85, CHAIN
DEF1 86, CLEAR
DEF3 87, CLG
DEF5 88, CLOSEIN
DEF7 89, CLOSEOUT
DEF9 8A, CLS
DEFB 8B, CONT
DEFD 8C, DATA
DEFF 8D, DEF
DF01 8E, DEFINT
DF03 8F, DEFREAL
DF04 90, DEFSTR
DF07 91, DEG
DF09 92, DELETE
DF0B 93, DIM
DF0D 94, DRAW
DF0F 95, DRAWR

DF11 96, EDIT
DF13 97, ELSE
DF15 98, END
DF17 99, ENT
DF19 9A, ENV
DF1B 9B, ERASE
DF1D 9C, ERROR
DF1F 9D, EVERY
DF21 9E, FOR
DF23 9F, GOSUB
DF25 A0, GOTO
DF27 A1, IF
DF29 A2, INK
DF2B A3, INPUT
DF2D A4, KEY
DF2F A5, LET
DF31 A6, LINE
DF33 A7, LIST
DF35 A8, LOAD
DF37 A9, LOCATE
DF39 AA, MEMORY
DF3B AB, MERGE
DF3D AC, MID$
DF3F AD, MODE
DF41 AE, MOVE
DF43 AF, MOVER
DF45 B0, NEXT
DF47 B1, NEW
DF49 B2, ON
DF4B B3, ON BREAK
DF4D B4, ON ERROR GOTO 0
DF4F B5, ON SQ

DF51 B6, OPENIN
DF53 B7, OPENOUT
DF55 B8, ORIGIN
DF57 B9, OUT
DF59 BA, PAPER
DF5B BB, PEN

DF5D BC, PLOT
DF5F BD, PLOTR
DF61 BE, POKE
DF63 BF, PRINT
DF65 C0, '
DF67 C1, RAD
DF69 C2, RANDOMIZE
DF6B C3, READ
DF6D C4, RELEASE
DF6F C5, REM
DF71 C6, RENUM
DF73 C7, RESTORE
DF75 C8, RESUME
DF77 C9, RETURN
DF79 CA, RUN
DF7B CB, SAVE
DF7D CC, SOUND
DF7F CD, SPEED
DF81 CE, STOP
DF83 CF, SYMBOL
DF85 D0, TAG
DF86 D1, TAGOFF
DF89 D2, TROFF
DF8B D3, TRON
DF8D D4, WAIT
DF8F D5, WEND
DF91 D6, WHILE
DF93 D7, WIDTH
DF95 D8, WINDOW
DF97 D9, WRITE
DF99 DA, ZONE
DF9B DB, DI
DF9D DC, EI
DF9E DD, FILL
DFA1 DE, GRAPHICS
DFA3 DF, MASK
DFA5 E0, FRAME
DFA7 E1, CURSOR
DFAA début de la RAM libre

DFB2   max. 300 caractères
DFB5   aller chercher caractère dans buffer d'entrée
DFB9   dernier caractère ?
DFBA   non
DFBE   301 - état compteur
DFC0   égale longueur de ligne
DFC3   dans B
DFC6   trois fois zéro comme terminaison
******   aller chercher caractère dans buffer d'entrée
DFCD
DFCE   dernier caractère ?
DFD0   lettre ?
DFD3   oui
DFD5   numerique ?
DFD8   oui
DFDB   '&' ?
DFDD   oui
DFE1   Token ?
DFE2   oui
DFE3   '!'
DFE8   ignorer espaces supplémentaires ?
E0A3   écrire dans buffer
E0B6   adresse de base de la table
E0B9   rechercher dans la table
******   instructions avec numéro de ligne
E0C8   'RESTORE'
E0C9   'AUTO'
E0CA   'RENUM'
E0CB   'DELETE'
E0CC   'EDIT'
E0CD   'RESUME'
E0CE   'ERL'
E0CF   'ELSE'
E0D0   'RUN'
E0D1   'LIST'
E0D2   'GOTO'
E0D3   'THEN'
E0D4   'GOSUB'
E0D5   fin de la table

E0D6 ’!’
E0DA ’&’
E0DD ’$’
E0F9 Token pour numéro de ligne
E105 tester si chaîne
E108 Token pour nombre à virgule flottante
E112 Token pour nombre deux octets
E119 10
E11D additionner Offset
E121 Token pour nombre un octet
E123 écrire dans buffer
E128 écrire dans buffer
E12F écrire dans buffer
E134 comparaison HL <> DE
E145 Token pour nombre binaire
E14B écrire dans buffer
E152 aller chercher type de variable
E158 écrire dans buffer
E161 ’"’
E165 ’|’, extension d’instruction
E16B ’?’
E16D Token pour ’PRINT’
E172 adresse des opérateurs BASIC
E187 ’’’
E18B écrire dans buffer
E190 ’"’
E19A écrire dans buffer
E1A1 ’"’
E1A5 écrire dans buffer
****** traiter extension d’instruction
E1A8 écrire dans buffer
E1AB zéro
E1AF écrire dans buffer
E1B2 prochain caractère
E1B3 incrémenter pointeur
E1B4 tester si lettre ou chiffre
E1B7 oui, alors dans buffer
E1BA pointeur de un en arrière
E1BC fixer bit 7 sur le dernier caractère

E1C3 écrire dans buffer
E1C6 "'
E1C8 écrire dans buffer
E1CB caractère
E1CE écrire dans buffer jusqu'à la fin de la ligne
****** instruction BASIC LIST
E1D2 aller chercher numéro de zone de ligne
E1D7 aller chercher numéro de canal
E1DA fin de l'instruction, sinon 'Syntax error'
E1DD adresse de ligne actuelle sur zéro
E1E2 lister lignes
E1E5 au mode READY
****** lister lignes BASIC BC- DE
E1E8
E1E9 numéro de ligne dans DE
E1EB chercher ligne BASIC DE
E1F0 fin du programme ?

E1F5 terminé
E1F6 interruption par 'ESC'
E1FA additionner longueur de ligne
E201 prochain numéro de ligne dans DE
E205 comparaison HL <> DE
E209 supérieur dernier numéro de ligne ?
E20B lister ligne BASIC dans buffer
E20E pointeur sur buffer
E211 sortir caractère
E214 incrémenter pointeur
E215 prochain caractère
E217 pas encore fin ?
E219 sortir LF
E21F lister prochaine ligne
E222 charger canal de sortie plus petit 8 ?
E225 caractère
E226 oui, sortie écran
E228 sortir caractère
E22B LF
E22E CR
E232 caractère de contrôle ?

E234 sortir comme caractère imprimable
E23A sortir caractère
E249 pointeur sur buffer
E26B token d'instruction ?
E27C sortir constante
E27E 'l', extension d'instruction
E290 '''
E294 'ELSE'
E299 ':'
E29D '"'
E2A8 '"'
E2CF écrire caractère dans buffer
E2D0 incrémenter pointeur de buffer
****** lister extension d'instruction
E2D6
E2D8 écrire dans buffer
E2DC prochain caractère
E2DD incrémenter pointeur
E2DE fin de ligne ?
E2DF non
E2E1 annuler bit 7
E2E3 écrire dans buffer
E2E6 dernier caractère ?
E2E8 non, prochain caractère
E2ED ' '
E2EF écrire dans buffer
E302 fonction ?
E324 tester si lettre ou chiffre
E337 nombre à virgule flottante ?

E33F nombre binaire ?
E343 nombre hexadécimal ?
E347 adresse de ligne ?
E34B numéro de ligne ?
E34F nombre deux octets ?
E356 nombre un octet ?
E35B chiffre ?
E376 'X'
E38D '&'

E398 type de variable 'Real'
E39A aller chercher nombre
E3AE 'A'
E3B3 plus #E41D, adresse des mots d'instruction
E3BF 26 lettres
E3C1 table des mots d'instruction
E3CA prochaine lettre
E3CC table des opérateurs de base
E3D2 'Syntax error'
E3F6 TAB
E3FA ' '
******* adresses des mots d'instruction
E41D A
E41F B
E421 C
E423 D
E425 E
E427 F
E429 G
E42B H
E42D I
E42F J
E431 K
E433 L
E435 M
E437 N
E439 O
E43B P
E43D Q
E43F R
E441 S
E443 T
E445 U
E447 V
E449 W
E44B X
E44D Y
E44F Z
****** table des instructions BASIC

****** lettre Z
E451 DA ZONE
****** lettre Y
E456 48 YPOS
****** lettre X
E45B 47 XPOS
E45F FD XOR
****** lettre W
E463 D9 WRITE

E468 D8 WINDOW
E46E D7 WIDTH
E473 D6 WHILE
E478 D5 WEND
E47C D4 WAIT
****** lettre V
E481 7F VPOS
E485 1D VAL
****** lettre U
E489 ED USING
E48E 1C UPPER$
E494 1B UNT
****** lettre T
E498 D3 TRON
E49C D2 TROFF
E4A1 EC TO
E4A4 46 TIME
E4A7 EB THEN
E4AB 7D TESTR
E4B0 7C TEST
E4B4 1A TAN
E4B7 D1 TAGOFF
E4BD D0 TAG
E4C0 EA TAB
****** lettre S
E4C4 CF SYMBOL
E4CA E7 SWAP
E4CE 7B STRING$
E4D5 19 STR$

E4D9 CE STOP
E4DD E6 STEP
E4E1 18 SQR
E4E3 17 SQ
E4E6 CD SPEED
E4EB E5 SPC
E4EE 16 SPACE$
E4F3 CC SOUND
E4F9 15 SIN
E4FC 14 SGN
E4FF CB SAVE
****** lettre R
E504 CA RUN
E507 7A ROUND
E50C 45 RND
E50F 79 RIGHT$
E515 C9 RETURN
E51B C8 RESUME
E521 C7 RESTORE
E528 C6 RENUM
E52D 13 REMAIN
E533 C5 REM
E536 C4 RELEASE
E53D C3 READ
E541 C2 RANDOMIZE
E549 C1 RAD
****** lettre Q
****** lettre P
E54F BF PRINT
E554 78 POS
E557 BE POKE
E55B BD PLOTR
E560 BC PLOT
E564 44 PI
E566 BB PEN
E569 12 PEEK
E56D BA PAPER
****** lettre O
E573 B9 OUT

E576 B8 ORIGIN
E57C FC OR
E57E B7 OPENOUT
E585 B6 OPENIN
E58B B5 ON SQ
E598 B4 ON ERROR GOTO 0
E5A0 B3 ON BREAK
E5A8 B2 ON
****** lettre N
E5AB FE NOT
E5AE B1 NEW
E5B1 B0 NEXT
****** lettre M
E5B6 AF MOVER
E5BB AE MOVE
E5BF AD MODE
E5C3 FB MOD
E5C6 77 MIN
E5C9 AC MID$
E5CD AB MERGE
E5D2 AA MEMORY
E5D8 76 MAX
E5DB DF MASK
****** lettre L
E5E0 11 LOWER$
E5E6 10 LOG10
E5EB 0F LOG
E5EE A9 LOCATE
E5F4 A8 LOAD
E5F8 A7 LIST
E5FC A6 LINE
E600 A5 LET
E603 0E LEN
E606 75 LEFT$
****** lettre K
E60C A4 KEY
****** lettre J
E610 0D JOY
****** lettre I

```
****** lettre I
E614   0C INT
E617   74 INSTR
E61C   A3 INPUT
E621   0B INP
E624   43 INKEY$
E62A   0A INKEY
E62F   A2 INK
E632   A1 IF
****** lettre H
E635   42 HIMEM
E63A   73 HEX$
****** lettre G
E63F   DE GRAPHICS
E647   A0 GO TO
E64C   9F GO SUB
****** lettre F
E653   09 FRE
E656   E0 FRAME
E65B   9E FOR
E65E   E4 FN
E660   08 FIX
E663   DD FILL
****** lettre E
E668   07 EXP
E66B   9D EVERY
E670   9C ERROR
E675   41 ERR
E678   E3 ERL
E67B   9B ERASE
E680   40 EOF
E683   9A ENV
E686   99 ENT
E689   98 END
E68C   97 ELSE
E690   DC EI
E692   96 EDIT
****** lettre D
E697   95 DRAWR
```

E69C 94 DRAW
E6A0 93 DIM
E6A3 DB DI
E6A5 49 DERR
E6A9 92 DELETE
E6AF 91 DEG
E6B2 90 DEFSTR
E6B8 8F DEFREAL
E6BF 8E DEFINT
E6C5 8D DEF
E6C8 72 DEC$
E6CC 8C DATA

****** lettre C
E6D1 E1 CURSOR
E6D7 06 CREAL
E6DC 05 COS
E6DF 7E COPYCHR$
E6E7 8B CONT
E6EB 8A CLS
E6EE 89 CLOSEOUT
E6F6 88 CLOSEIN
E6FD 87 CLG
E700 86 CLEAR
E705 04 CINT
E709 03 CHR$
E70D 85 CHAIN
E712 84 CAT
E715 83 CALL
****** lettre B
E71A 82 BORDER
E720 71 BIN$
****** lettre A
E725 81 AUTO
E729 02 ATN
E72C 01 ASC
E72F FA AND
E732 80 AFTER
E737 00 ABS

****** opérateurs BASIC et Tokens correspondants
E73B F8 '^'
E73D F9 'Backslash'
E740 F0 '>='
E743 F0 '=>'
E747 EE '>'
E749 F2 '<>'
E74D F3 '<='
E751 F3 '=<'
E755 EF '='
E757 F1 '<'
E759 F7 '/'
E75B 01 ':'
E75D F6 '*'
E75F F5 '-'
E761 F4 '+'
E763 C0 '''
****** annuler pointeur de programme
E766
E767 début du programme
E76D trois fois zéro à la fin du programme
E770 fin du programme
E77D placer numéro de ligne
****** remplacer adresses de ligne par numéro de ligne
E78B aller chercher prochain élément de la ligne
E78E fin de l'instruction ?
E790 oui
E791 'adresse de ligne' ?
E793 non

E79F numéro de ligne dans DE
E7A2 'numéro de ligne'
E7A6 placer
****** instruction BASIC DELETE
E7F3
E7F6 fin de l'instruction, sinon 'Syntax error'
E802 au mode READY
E805 aller chercher numéro de zone de ligne
E80A chercher ligne BASIC DE

E80F chercher ligne BASIC DE
****** aller chercher adresse de ligne
E82C
E82E numéro ou adresse dans DE
E830 'adresse de ligne' ?
E832 oui
E834 'numéro de ligne' ?
E836 'Syntax error'
E83A aller chercher numéro de ligne dans HL
E83D comparaison HL <> DE
E840 plus petit, chercher à partir du début du programme
E845 ignorer reste de la ligne
E848 à partir de adresse (HL)
E849 chercher ligne BASIC DE
E84C pas trouvé, chercher à partir de début du programme
E854 'adresse de ligne'
E859 dans le programme
E85B adresse de ligne dans le programme
****** chercher ligne BASIC DE
E861
E865 sortir message d'erreur
E868 'Line does not exist'
E869 début du programme
E86E longueur de ligne dans BC
E872 fin du programme ?
E873 pas trouvé
E878 numéro de ligne dans HL
E87C comparaison HL <> DE
E882 supérieur, pas trouvé
E883 égal, trouvé
E884 additionner longueur de ligne
E885 chercher encore
****** chercher ligne BASIC DE
E887 début du programme
E88B ranger adresse de ligne
E88D longueur de ligne dans BC
E891 fin du programme ?
E893 oui
E896 numéro de ligne dans HL

E89A comparaison HL <> DE
E89F actuel numéro de ligne supérieur ou égal ?
E8A0 additionner longueur de ligne
E8A1 chercher encore
****** instruction BASIC RENUM
E8A3 10, défaut pour valeur initiale
E8A6 aller chercher numéro de ligne dans DE
E8AA 0, défaut
E8AD suit virgule ?
E8B0 aller chercher numéro de ligne dans DE
E8B4 10, défaut
E8B7 suit virgule ?
E8BD fin de ligne, sinon 'Syntax error'
E8C6 chercher ligne BASIC DE
E8CB chercher ligne BASIC DE
E8D0 comparaison HL <> DE
E8D3 'Improper argument'

E8F2 'Improper argument'
E967 'IF'
E974 'ELSE'
E980 ']'
E984 '('
E98D ']'
E98F
E991 '('
E995 '['
E999 ')'
E9A1 'Syntax error'
****** instruction BASIC DATA
E9A8
****** instructions BASIC REM et '
E9AC
****** instruction BASIC ELSE
E9B2
E9C2 début du programme
E9D1 saut en (BC)
E9EC sortir message d'erreur
E9FA 'ELSE'

E9FD 'THEN'
EA02 ignorer les espaces
EA0E '"'

EA12 'l'
EA16 '''
EA1A 'REM'
EA1E fonction
EA25 '"'
EA2C '"'
****** instruction BASIC RUN
EA7D fin de l'instruction ?
EA80 début du programme comme défaut
EA84 oui
EA86 numéro de ligne ?
EA88 oui
EA8A adresse de ligne ?
EA94 MC BOOT PROGRAMM
EA9A début du programme
EA9F aller chercher adresse de ligne
EAB7 à la boucle de l'interpréteur
****** instruction BASIC LOAD
EABA
EAC2 au mode READY
EACC DISK IN DIRECT
EAD6 aller chercher nom, ouvrir fichier
EAD9 type de fichier
EAE7 suit virgule ?
EAEA oui, aller chercher valeur 16 bits
EAED comme adresse de début
EAF1 fin de l'instruction, sinon 'Syntax error'
EAF6 adresse de début
EAF9 DISK IN DIRECT
EAFD DISK IN CLOSE
****** instruction BASIC CHAIN
EB02 'MERGE'
EB04 flag pour MERGE
EB07 ignorer les espaces
EB0D valeur défaut zéro pour ligne de début

EB10 suit virgule ?
EB13 non
EB16 ','
EB18 aller chercher valeur 16 bits
EB1C suit virgule ?
EB1F non
EB21 tester si encore un caractère
EB24 'DELETE'
EB25 annuler zone de ligne
EB2A fin de l'instruction, sinon 'Syntax error'
EB30 Garbage Collection
EB3D aller chercher ligne de début
EB3E début du programme comme défaut
EB42 aucune ligne de début
EB4C flag pour MERGE
****** instruction BASIC MERGE
EB59 aller chercher nom, ouvrir fichier
EB5C fin de l'instruction, sinon 'Syntax error'
EB5F annuler variables
EB62 tester type de fichier
EB65 au mode READY
EB71 fin du programme
EB75 début du programme
EB79 fin du programme
EB7D BD := HL - DE

EBA5 comparaison HL <> DE
EBBA fin du programme
EBCD comparaison HL <> DE
EBE9 fin du programme
EBF1 fin du programme
EBFD 'Memory full'
EBFF sortir message d'erreur
EC01 DISK IN CHAR
EC05 CTRL Z
EC09 erreur disquette
EC0E erreur disquette
EC14 'EOF met'
EC1B sortir message d'erreur

EC24 fin du programme
EC2A fin du programme
EC32 fin du programme
EC4B fin du programme
EC67 type de fichier
EC6E fichier ASCII ?
EC70 non
EC72 type de fichier
EC75 fichier ASCII
EC77 oui
EC79 annuler bit 0 (fichier protégé)
EC7D sortir message d'erreur
EC80 'File type error'
EC87 début du programme
EC9A comparaison HL <> DE
ECA2 fin du programme
ECA5 type de fichier
ECA8 tester bit 0
ECAA fixer flag pour fichier protégé
ECAF DISK IN DIRECT
ECB2 'EOF met'
ECD8 'Direct command found'
ECDC 'Overflow'
ECDE sortir message d'erreur
****** instruction BASIC SAVE
ECE1
ECE4 OPENOUT
ECE7 type de fichier 0, programme BASIC
ECE9 suit virgule ?
ECEC non
ECEE tester si encore un caractère
ECF1 variable normale
ECF4 nom de variable
ECF5 convertir minuscules en majuscules
ECF7 'Syntax error'
ECFB adresse de base de la table
ECFE rechercher dans la table
ED01 adresse dans table sur pile
ED02 ignorer les espaces

ED05 nombre des entrées
ED06 pas trouvé, 'Syntax error'
ED08 'A'
ED0B 'B'
ED0E 'P'

****** SAVE ,P
ED11 type de fichier 1, protected
ED13 fin de l'instruction, sinon 'Syntax error'
ED1E début du programme
ED23 fin du programme
ED27 HL := HL - DE
****** SAVE ,B
ED30 tester si ','
ED33 aller chercher valeur 16 bits
ED36 ranger
ED37 tester si ','
ED3A aller chercher valeur 16 bits
ED3D ranger
ED3E suit virgule ?
ED41 0, défaut pour adresse d'entrée
ED44 oui, aller chercher valeur 16 bits
ED47 ranger
ED48 fin de l'instruction, sinon 'Syntax error'
ED4B type de fichier 2, binaire
ED4D adresse d'entrée
ED4E adresse de fin
ED4F adresse de début
ED50 DISK OUT DIRECT
ED53 interruption par 'ESC'
ED56 CLOSEOUT
****** SAVE ,A
ED58 fin de l'instruction, sinon 'Syntax error'
ED5C 9
ED5E sortie sur canal 9, disquette
ED62 1 à
ED65 65535
ED68 lister lignes
ED6C sortie à nouveau sur défaut

ED6F CLOSEOUT
ED79 annuler espace, TAB et LF
ED7E '&'
ED82 tester si numerique
ED87 type sur Integer
ED8A annuler variable
ED9A '&'
EDA3 accepter nombre entier dans HL
EDB0 '.'
EDB2 annuler espace, TAB et LF
EDB5 tester si chiffre
EDBC '.'
EDC1 type sur Integer
EDCE '.'
EDD5 type sur Real
EDEF tester si chaîne
EDF2 non
EDFF Integer 4 octets * 256 en virgule flottante
EE03 multiplier nombre par 10^A
EE07 fixer type de variable sur virgule flottante
EE0A copier variable de (DE) dans (HL)
EE14 annuler espace, TAB et LF
EE18 -1
EE1A '-'
EE1D 0
EE1E '+'
EE24 annuler espace, TAB et LF
EE28 tester si chiffre
EE2E convertir minuscules en majuscules
EE33 '0'
EE47 'E'
EE52 annuler espace, TAB et LF
EE55 tester si chiffre
EE60 fixer type sur Real
EE6B 100

EE99 ignorer espace, TAB et LF
EEC0 fois 10
EEC4 plus prochain chiffre

EEDA accepter nombre entier dans HL
EEED ignorer espace, TAB et LF
EEF0 convertir minuscules en majuscules
EEF3 base 2, binaire
EEF5 'X'
EEF9 base 16, hex
EEFB 'H'
EF00 ignorer espace, TAB et LF
EF05 Basis 10, décimal
EF08 convertir chiffre (hexa) en binaire
EF12 convertir chiffre (hexa) en binaire
EF1C base du système numérique
EF1D multiplication entière sans signe
******* convertir chiffre (hexa) en binaire
EF31 aller chercher caractère
EF32 incrémenter pointeur
EF33 tester si chiffre
EF36 oui
EF38 convertir minuscules en majuscules
EF3B 'A'
EF3E plus petit que 'A', erreur
EF40 'A'-('9'+1)
EF42 '0'
EF45 pas erreur
EF47 annuler Carry
****** sortir nombre entier HL
EF49 convertir nombre entier en ASCII
EF4C sortir chaîne
******* convertir nombre entier en ASCII
EF4F
EF51 accepter nombre entier dans HL
EF61 zéro
EF62 convertir nombre en chaîne formatée
EF68 ' '
EF98 '%'

F02F '.'
F034 '1'
F03D '+E'

F047 '-'
F04C '0'-1
F04F 10
F053 '9'+1
F079 '.'
F099 '0'
F0A8 '5'
F0B1 '1'
F0C0 '9'
F0C3 '0'
F0DA '0'
F0E8 '0'
F128 ','
F135 '0'
F146 '0'
F156 '-'
F162 '+'
F166 ' '
F181 '*'
F185 ' '
F1CF '0'
F1DE '0'
****** fonction BASIC PEEK
F20D UNT
F210 READ RAM, LD A,(HL)
F211 accepter contenu accu comme nombre entier
****** instruction BASIC POKE
F214 aller chercher adresse 16 bits
F218 tester virgule et aller chercher valeur 8 bits
F21C écrire valeur dans adresse
****** fonction BASIC INP
F21E CINT
F221 adresse de port dans BC
F223 lire port
F225 accepter contenu accu comme nombre entier
****** instruction BASIC OUT
F228 aller chercher adresse et valeur
F22B sortir
****** instruction BASIC WAIT

F22E aller chercher adresse et valeur

F231 valeur 8 bits dans D

F232 3ème paramètre zéro

F234 aller chercher évent. troisième paramètre

F237 troisième paramètre dans E

F238 lire port

F23B relier

F23C et attendre

****** aller chercher valeurs 16 bits et 8 bits

F23F aller chercher valeur 16 bits

F243 dans BC

F244 tester si ','

F247 et aller chercher valeur 8 bits

****** extension d'instruction

F24A incrémenter pointeur de programme

F24C suit octet zéro ?

F24F oui, KL FIND COMMAND

F252 adresse d'instruction dans DE

F254 pas trouvé, 'Unknown command'

F257 ignorer mot d'instruction

F257 ignorer mot d'instruction

F258 bit 7 mis ?

F259 non, continuer à lire

F25B à l'instruction CALL

F25D sortir message d'erreur

F260 'Unknown command'

****** instruction BASIC CALL

F261 aller chercher valeur 16 bits

F264 #FF = RAM sélectionnée

F266 adresse dans #AE55

F26B octet de configuration dans #AE57

F26E sauver pointeur de pile

F272 maximum 32 paramètres

F274 suit virgule ?

F277 non

F27A aller chercher expression

F27E et adresse sur pile

F27F prochain paramètre

F281 fin de l'instruction, sinon 'Syntax error'

F284 sauver HL
F289 nombre des paramètres dans accu
F28E adresse du bloc de paramètres dans IX
F290 exécuter routine
F293 ramener pointeur de pile
F297 initialiser descripteur de pile
F29A restaurer HL
****** instruction BASIC ZONE
F2A2 aller chercher valeur 8 bits différente de zéro
F2A5 comme pas de tabulation
****** instruction BASIC PRINT
F2A9 numéro de canal
F2AC fin de l'instruction ?
F2AF oui, sortir LF
F2B2 'USING'
F2B8 adresse de base de la table
F2BB rechercher dans la table
F2BF JP (DE), exécuter fonction
F2C2 fin de l'instruction ?
F2C5 non, continuer
F2C8 nombre des entrées de la table
F2C9 adresse de retour si pas trouvé
F2CB ','
F2CE 'SPC'
F2D1 'TAB'
F2D4 ';'
F2D5 ignorer espaces
****** PRINT
F2D7 aller chercher expression
F2DC tester si chaîne
F2DF oui
F2E1 convertir nombre en chaîne ASCII
F2E4 aller chercher param. de chaîne
F2E7 ajouter ' ' caractère espace
F2E9 aller chercher descripteur de chaîne
F2EC incrémenter longueur
F2F0 aller chercher descripteur de chaîne
F2F3 longueur
F2FF ' '

F302 caractère de contrôle ?
F30F canal de sortie sélectionné
****** PRINT ,
F31E ignorer les espaces
F321 pas de tabulation
****** PRINT SPC
F339 aller chercher valeur 8 bits dans parenthèses
****** PRINT TAB
F342 aller chercher valeur 8 bits dans parenthèses
****** aller chercher valeur 8 bits dans parenthèses
F362 ignorer les espaces
F365 tester si '('
F368 aller chercher valeur 8 bits
F36B tester si ')'

****** PRINT USING
F383 ignorer les espaces
F386 aller chercher expression chaîne
F389 tester si encore un caractère
F38C ';'
F392 aller chercher expression
F3A9 fin de l'instruction ?
F3AC oui
F3B4 aller chercher expression
F3D7 'Underline'
F3F4 ';'
F3F6 ignorer les espaces
F3F9 tester si ','
F3FF '&'
F404 '!'
F408 'Backslash'
F413 'Backslash'
F417 ' '
F436 tester si caractère de formatage
F443 formater nombre
F446 sortir chaîne
****** tester si caractère de formatage
F44D
F454 '+'

F460 '.'
F464 '#'
F47A ' '
F47C '*'
F489 '#'
F49C '$'
F4B0 'Improper argument'
F4B8 '.'
F4BC '#'
F4C0 ','
F4D0 '#'
F4D6 '^'
F4F9 '-'
F4FD '+'
F507 '$'
****** instruction BASIC WRITE
F50D
F510 fin de l'instruction
F513 oui
F516 aller chercher expression
F51B tester si chaîne
F51E oui
F520 convertir nombre en ASCII
F523 et sortir
F528 '"'
F52A sortir
F52D sortir chaîne
F530 '"'
F532 sortir
F537 fin de l'instruction ?
F53D ','
F53F sortir
F542 continuer
****** configurer mémoire
F544 place mémoire de DE à HL
F547 comparaison HL avec BC
F54A plus grande adresse < #AC00 ?
F54B HIMEM
F54E fin des chaînes

F551 fin de la RAM libre
F555 début de la RAM libre
F558 plus 303
F55D donne début du programme
******* instruction BASIC MEMORY
F570 aller chercher valeur 16 bits
F577 comparaison HL <> DE
****
F58F TXT GET M TABLE
F5F7 comparaison HL <> DE
******* calculer longueur de la zone des chaînes
F5FD
F5FF début des chaînes
F603 fin des chaînes
F606 BC := HL - DE
******* incrémenter pointeurs progr. et de variable de BC
F60C fin du programme
F610 fin du programme
F618 début des variables
F61C début des variables
F61F début des tableaux
F623 début des tableaux
F626 fin des tableaux
F62A fin des tableaux
F633 fin du programme
F63E fin du programme
F645 BC := HL - DE
****** initialiser pile BASIC
F652 début de la pile
F655 pointeur de pile BASIC
F658 pour un octet
F65A reserver place dans pile BASIC
F65D zéro sur pile
F65F incrémenter pointeur de pile
F660 et ranger
****** libérer place dans pile BASIC
F665 pointeur de pile
F669 retrancher contenu accu
F671 ranger nouvelle valeur du pointeur de pile BASIC

****** reserver place dans pile BASIC
F675 pointeur de pile BASIC
F67A additionner contenu accu
F67E pointeur de pile BASIC
F683 donne plus de #4F94 de dépassement ?
F686 alors pointeur de pile est > #B06C
F689 initialiser pile BASIC
F68C 'Memory full'
F68F fin des chaînes
F692 début des chaînes
******* reserver place pour chaîne
F696
F69C début des chaînes
F6A4 comparaison HL <> DE
F6AE sortir message d'erreur
F6B1 'chaîne space full'
F6B2 début des chaînes
F6BF fin du programme
F6D2 comparaison HL <> DE
F6EA transfert de bloc LDDR
F6F9 BC := HL - DE
F6FE transfert de bloc LDIR
F705 BC := HL - D
F70C début des chaînes
F717 début des chaînes
****** instruction BASIC SYMBOL
F784 'AFTER'
F788 aller chercher valeur 8 bits
F78C tester si ','
F78F 8 valeurs
F792 suit virgule ?
F796 oui, aller chercher valeur 8 bits

F79B déjà 8 arguments ?
F79F TXT GET MATRIX
F7A2 matrice pas dans RAM, 'Improper argument'
F7A5 8
F7A8 plus adresse matrice
F7A9 un octet de la pile

F7AB dans table matrice
F7AD prochain octet
****** SYMBOL AFTER
F7B1 ignorer les espaces
F7B4 aller chercher valeur entière avec signe
F7B8 256
F7BB comparaison HL <> DE
F7BE 'Improper argument'
F7C2 TXT GET M TABLE
F7C6 matrice pas encore défini ?
F7D3 'Improper argument'
F7DD 256
F7E0 TXT SET M TABLE
F7FD 'Memory full'
F805 TXT SET M TABLE
F815 comparaison HL <> DE
F818 'Memory full'
F833 fin des chaînes
F83E transfert de bloc LDDR
F844 début des chaînes
F851 transfert de bloc LDIR
F857 fin des chaînes
F85B début des chaînes
F865 fin du programme
F868 comparaison HL <> DE
F875 sortir message d'erreur
F878 'Memory full'
****** lire chaîne
F879
F87E '"'
F880 ignorer les espaces
F89F ','
F8AD JP (DE)
F8BE ' '
F8C2 TAB
F8C6 CR
F8CA LF
****** sortir chaîne
F8D0 aller chercher param. de chaîne

F8D3 chaîne vide ?
F8D4 aller chercher caractère
F8D5 incrémenter pointeur
F8D6 sortir caractère
F8D9 prochain caractère
****** fonction BASIC LOWER$
F8EC convertir majuscules en minuscules
****** convertir majuscules en minuscules
F8F1 'A'
F8F4 'Z'+1
F8F7 'a'-'A'
****** fonction BASIC UPPER$
F8FA convertir minuscules en majuscules
F915 saut en (BC)
****** addition de chaîne
F91D pointeur sur seconde chaîne
F921 longueurs
F922 additionner
F923 pas dépassement
F925 sortir message d'erreur
F928 'String too long'
****** fonction BASIC BIN$
F964
****** fonction BASIC HEX$
F969
F96D aller chercher expression
F975 suit virgule ?
F978 0 comme défaut
F979 oui, aller chercher valeur 8 bits
F97C supérieur ou égal 17 ?
F97E oui, 'Improper argument'
F982 tester si ')'
F98A convertir nombre en chaîne
******* fonction BASIC DEC$
F98F aller chercher expression
F992 tester si ','
F995 placer sur pile BASIC
F998 aller chercher expression chaîne et paramètre
F99B tester si ')'

F99F longueur
F9A0 libérer place dans pile BASIC
F9A4 longueur
F9A5 accepter variable
F9AB tester si caractère de formatage
F9AE 'Improper argument'
F9B3 'Improper argument'
F9B7 formater nombre
F9BA accepter chaîne
****** fonction BASIC STR$
F9BC
F9BD convertir nombre en chaîne
F9C1 compteur pour longueur de chaîne sur -1
F9C3 zéro
F9C4 incrémenter compteur
F9C5 fin des chaînes ?
F9C6 incrémenter pointeur
F9C7 non, prochain caractère
F9CA longueur de chaîne
F9CB réserver place, créer descripteur de chaîne
****** fonction BASIC LEFT$
F9D3 aller chercher chaîne et valeur 8 bits
****** fonction BASIC RIGHT$
F9D8 aller chercher chaîne et valeur 8 bits
F9DB longueur de chaîne
F9DC moins paramètre
****** fonction BASIC MID$
F9E2 tester si '('
F9E5 aller chercher chaîne et valeur 8 bits
F9E8 zéro, 'Improper argument'
F9EB 255
F9EC comme défaut
F9ED aller chercher troisième argument
F9F0 tester si ')'
****** instruction BASIC MID$
FA07 tester si '('
FA0A aller chercher variable
FA0D type chaîne, sinon 'Type mismatch'
FA19 'Improper argument'

FA1C 255
FA1D comme défaut
FA1E aller chercher troisième argument
FA21 tester si ')'
FA24 tester si '='
FA28 aller chercher expression et paramètre chaîne
FA3E transfert de bloc LDIR
FA43 aller chercher expression chaîne
****** aller chercher troisième argument pour MID$
FA4F défaut 255
FA52 ')'

FA56 tester si ','
FA59 aller chercher valeur 8 bits
****** fonction BASIC LEN
FA69 aller chercher param. de chaîne, longueur dans A
FA6C accepter contenu accu comme nombre entier
****** fonction BASIC ASC
FA6E code ASCII du premier caractère
FA71 accepter contenu accu comme nombre entier
****** fonction BASIC CHR$
FA74 CINT, <256
FA77 code ASCII dans accu
FA7A longueur 1
FA7C créer chaîne avec longueur A
****** fonction BASIC INKEY$
FA7E KM READ CHR
FA81 aucune touche appuyée ?
FA83 'ESC'
FA85 chaîne vide
FA87 'ESC
FA89 chaîne vide
FA8B accepter caractère dans chaîne
****** fonction BASIC STRING$
FA8D aller chercher valeur 8 bits, longueur
FA91 tester si ','
FA94 aller chercher expression
FA97 tester si ')'
FA9F créer chaîne avec longueur A

FAA1 tester si chaîne
FAA4 non
FAA6 aller chercher param. de chaîne
FAA9 chaîne vide, 'Improper argument'
FAAB aller chercher code ASCII
****** fonction BASIC SPACE$
FAAD
FAB0 ' '
****** fonction BASIC VAL
FABE aller chercher param. de chaîne
FAC1 accepter contenu accu comme nombre entier
FACE convertir chaîne en nombre
FAD5 sortir message d'erreur
FAD8 'type mismatch'
FAE2 'Improper argument'
****** fonction BASIC INSTR
FAE5 aller chercher expression
FAE8 tester si chaîne
FAEB position début défaut 1
FAED oui
FAEF CINT, < 256
FAF3 'Improper argument'
FAF7 tester si ','
FAFA aller chercher expression chaîne
FAFD tester si ','
FB05 aller chercher expression et paramètre chaîne
FB08 tester si ')'
FB48 accepter contenu accu comme nombre entier
FB65 début des chaînes
FB68 comparaison HL <> BC
FB6D fin des chaînes
FB70 comparaison HL <> BC
FB9E début du programme
FBA1 comparaison HL <> DE
FBA4 fin des chaînes
FBA8 comparaison HL <> DE
FBAD fin du programme
FBB1 comparaison HL <> DE
FBC4 transfert de bloc LDIR

******* initialiser descripteur de pile
FBCC
FBCF pointeur dans descripteur de pile pour chaînes
FBD7 chaîne
FBD9 comme type de variable
FBDC pointeur dans descripteur de pile
FBE2 descripteur de chaîne
FBE5 comparaison HL <> DE
FBE8 'String expression too complex'
FBEA sortir message d'erreur
FBF0 pointeur dans descripteur de pile
FBF6 type chaîne, sinon 'type mismatch'
FC0A début des chaînes
FC10 comparaison HL <> DE
FC1B début des chaînes
FC27 comparaison HL <> DE
****** fonction BASIC FRE
FC53 tester si chaîne
FC56 non
FC5B Garbage Collection
FC5E calculer place mémoire libre
****** Garbage Collection
FC64
FC7C comparaison HL <> DE
FC87 fin des chaînes
FC8B début des chaînes
FCA9 comparaison HL <> BC
FCAF début des chaînes
FCB3 BC := HL - DE
FCB7 comparaison HL <> DE
FCBC transfert de bloc LDDR
FCC0 début des chaînes
FCD9 comparaison HL <> DE
FCE6 comparaison HL <> DE

FCF3 aller chercher résultat numérique
FD03 UNT
******* opérateur BASIC '+'
FD0C tester type des opérandes

FD0F virgule flottante ?
FD11 addition entière HL := HL + DE
FD14 pas dépassement, accepter résultat dans HL
FD17 convertir en virgule flottante
FD1A addition à virgule flottante
FD1D pas dépassement, ok
FD1E 'Overflow'
******* opérateur BASIC '-'
FD21 tester type des opérandes
FD24 virgule flottante ?
FD26 soustraction entière HL := DE - HL
FD29 pas dépassement, accepter résultat dans HL
FD2C convertir en virgule flottante
FD2F soustraction à virgule flottante
FD32 pas dépassement, ok
FD33 'Overflow'
******* opérateur BASIC '*'
FD35 tester type des opérandes
FD38 virgule flottante ?
FD3A multiplication entière avec signe
FD3D pas dépassement, accepter résultat dans HL
FD40 convertir en virgule flottante
FD43 multiplication à virgule flottante
FD46 pas dépassement, ok
FD47 'Overflow'
****** comparaison arithmétique
FD49 tester type des opérandes
FD4C comparaison entiers
FD4F comparaison à virgule flottante
****** opérateur BASIC '/'
FD52
FD57 division à virgule flottante
FD5B 5 octets
FD5E transférer résultat
FD60 ok ?
FD61 'Division by zero'
FD64 'Overflow'
******* opérateur BASIC 'Backslash'
FD67

FD6B   division entière avec signe
FD6E   accepter résultat dans HL
FD71   'Division by zero'
******* opérateur BASIC 'MOD'
FD79
FD7D   calcul MOD
FD80   accepter résultat dans HL
FD83   sortir message d'erreur
FD86   'Division by zero'
******* opérateur BASIC 'AND'
FD87
FD8C   HL := HL AND DE
FD8F   accepter nombre entier HL
******* opérateur BASIC 'OR'
FD92
FD97   HL := HL OR DE
FD9A   accepter nombre entier HL
******* opérateur BASIC 'XOR'
FD9C
FDA1   HL := HL XOR DE
FDA4   accepter nombre entier HL
****** opérateur BASIC 'NOT'
FDA6   CINT
FDAC   former complément de HL
FDAE   accepter nombre entier HL
******* fonction BASIC ABS
FDB0   SGN
FDB3   signe positif, terminé
****** inverser signe
FDB4   aller chercher résultat numérique
FDB7   changement de signe virgule flottante
FDBA   changement de signe Integer
FDBD   sauver résultat
FDC0   dépassement, convertir nombre en virgule flottante
******* déterminer signe
FDC4
FDC6   SGN
******* déterminer signe
FDCC   aller chercher résultat numérique

FDCF Integer SGN
FDD2 SGN
******* arrondir nombre
FDD7 accepter type et valeur de variable
FDD7 accepter type et valeur de variable
FDDB aller chercher résultat numérique
FDDE chiffres d'arrondissage
FDDF valeur à virgule flottante ?
FDE2 arrondissage après virgule? terminé
FDE3 convertir valeur entière en virgule flottante
FDE6 arrondir nombre
FDE9 CINT
FDEC chiffres d'arrondissage
FDED differ. de zéro, alors arrondir
FDEF convertir virgule flottante en Integer
FDF2 exécuter fonction
FDF6 chiffres d'arrondissage
FDF7 multiplier nombre à virgule flottante par 10^A
FDFA convertir virgule flottante en Integer
FE02 convertir Integer en virgule flottante
FE05 inverser chiffres d'arrondissage
FE06 correspond à division
FE07 multiplier nombre à virgule flottante par 10^A
****** fonction BASIC FIX
FE0E fonction FIX
****** fonction BASIC INT
FE13 fonction INT
FE16 aller chercher résultat numérique
FE19 Integer ?
FE1A JP (DE), exécuter fonction
FE1E type de variable
FE25 type de variable dans C, pointeur dans HL
FE29 convertir Integer en virgule flottante
FE2D chaîne ?
FE34 si positif accepter signe de B
FE38 accepter résultat dans HL
FE3C chaîne ?
FE3E oui, 'type mismatch'
FE40 type de variable

FE43 chaîne ?
FE45 oui, 'type mismatch'
FE4D type de variable
FE50 ranger
FE52 convertir nombre entier en virgule flottante
FE58 convertir nombre entier en virgule flottante
FE5D pointeur sur variable
FE68 variable
FE6D 'type mismatch'
FE70 type de variable
FE73 comparer
FE74 Integer ?
FE76 non

******* opérande Integer en virgule flottante
FE78 premier opérande
FE7B convertir
FE7E pointeur de pile BASIC, second opérande
FE81 convertir
FE84 dans DE
******* convertir nombre entier en virgule flottante
FE8D nombre dans DE
****** convertir nombre entier en virgule flottante
FE95 type de variable
FE98 sur 'Real'
FE9E négatif, alors changement de signe integer
FEA2 convertir Integer en virgule flottante
******* convertir nombre 4 octets en virgule flottante
FEA5 Lo-Word
FEA9 Hi-Word
FEAC type de variable
FEAF 'Real'
FEB1 pointeur sur valeur 4 octets
FEB3 convertir nombre en virgule flottante
****** fonction BASIC CINT
FEB6
FEBA 'Overflow'
FEBF résultat
FECC 'Overflow'

FECE pointeur sur type de variable
FED1 charger type de variable
FED2 type sur Integer
FED5 comparer avec chaîne
FED9 'type mismatch'
FEDD convertir nombre à virgule flottante en Integer
FEE1 accepter signe B dans nombre entier HL
****** valeur entière (HL) dans HL
FEE6
****** fonction BASIC UNT
FEEB aller chercher résultat numérique
FEEE Integer ?
FEEF convertir virgule flottante en Integer
FEF2 'Overflow'
FEF5 accepter signe B dans nombre entier
FEF8 accepter nombre entier dans HL
FEFB sortir message d'erreur
FEFE 'Overflow'
FF02 type de variable
FF05 comparer
FF06 différent ?
FF0F CINT
FF11 type chaîne, sinon 'Type mismatch'
****** fonction BASIC CREAL
FF14 aller chercher résultat numérique
FF17 Integer, alors convertir
******* fixer valeur à virgule flottante sur zéro
FF1B
****** fonction BASIC SGN
FF2A SGN
****** accepter contenu accu comme nombre entier
FF32 octet faible
FF33 annuler octet fort
****** accepter nombre entier dans HL
FF35 ranger valeur
FF38 type sur Integer
FF3A et ranger
****** type de variable sur virgule flottante
FF3E pointeur sur nombre à virgule flottante

FF41 type sur Real
****** aller chercher type de variable, HL pointe sur variable
FF45 pointeur sur variable
FF48 type dans C
FF49 HL pointe sur variable
****** aller chercher type de variable
FF4B type de variable dans accu
****** aller chercher résultat numérique
FF4F type de variable
FF52 chaîne ?
FF54 oui, 'type mismatch'
FF56 charger valeur entière
FF59 pas virgule flottante, terminé
FF5A adresse du nombre à virgule flottante
FF5E tester si chaîne
FF61 oui, ok
FF62 sortir message d'erreur
FF65 'type mismatch'
****** tester si chaîne
FF66 type de variable
FF69 chaîne ?
FF6C fixer type de variable
FF6F adresse dans DE
****** placer résultat sur pile BASIC
FF74
FF76 type de variable
FF79 égale besoin de pile
FF7A reserver place dans pile BASIC
FF7D placer résultat sur pile
****** copier variable dans (HL)
FF83 adresse objet dans DE
FF84 adresse source
FF88 type de variable
FF8B égale compteur de décalage
FF8C annuler octet fort
FF8E décaler
****** tester si lettres
FF92 convertir minuscules en majuscules
FF95 'A'

FF99 'Z'+1
******* tester si caractère alphanumérique
FF9C tester si lettre
FF9F oui
FFA0 '.'
FFA4 '0'
FFA8 '9'+1
****** convertir minuscules en majuscules
FFAB 'a'
FFAE 'z'+1
FFB1 'a'-'A'
****** rechercher dans table suivante
FFB4
FFB6 charger longueur de table
FFB8 adresse de retour pour recherche négative
FFBB comparer caractère
FFBC incrémenter pointeur
FFBD trouvé ?
FFBF table pas encore terminée ?
FFC1 charger adresse de retour
FFC5 adresse dans HL
****** rechercher dans zone de mémoire (HL)
FFCA
FFCC A dans C
FFCE zéro
FFD2 égale accu antérieur
FFD4 fixer Carry
******* comparaison HL <> DE
FFD8
FFD9 H - D
FFDB L - E
****** comparaison HL <> BC
FFDE
FFDF H - B
FFE1 L - C
******* BC := HL - DE
FFE4
FFE6 HL := HL - DE
FFE9 BC := HL

FFEC nombre dans C
FFED octet fort sur zéro
FFF0 compteur BC = 0 ?
FFF1 oui, alors terminé
FFF2 transfert de bloc
****** transfert de bloc LDDR
FFF5
FFF6 compteur BC = 0 ?
FFF7 oui, alors terminé
FFF8 transfert de bloc
****** saut en (HL)
FFFB
****** saut en (BC)
FFFC
****** saut en (DE)
FFFE

4.1 Les routines du système d'exploitation

Voici une liste des routines et tables du système d'exploitation, pour autant que nous les connaissions.

Attention: n'essayez jamais d'appeler ces routines à travers les adresses qui vous sont fournies ici si vous ne maîtrisez pas pleinement le mécanisme de commutation de la configuration mémoire!

Utilisez plutôt les vecteurs présentés au chapitre 2.1.

Cette liste sert avant tout à vous permettre d'avoir un rapide aperçu du système d'exploitation. C'est pour cela que nous n'avons présenté ici que les routines du CPC6128 (voir chapitre 2.5). Pour le CPC664, la liste correspondante différerait légèrement pour certaines adresses.

KERNAL

0000	RST 0 RESET ENTRY
0008	RST 1 LOW JUMP
0010	RST 2 SIDE CALL
0018	RST 3 FAR CALL
0020	RST 4 RAM LAM
0028	RST 5 FIRM JUMP
0030	RST 6 USER RESTART
0038	RST 7 INTERRUPT ENTRY
0040	jusqu'ici on copie dans la RAM
0044	Restore High Kernel Jumps
005C	KL CHOKE OFF
0099	KL TIME PLEASE
00A3	KL TIME SET
00B1	Scan Events
0153	Kick Event
0163	KL NEW FRAME FLY
016A	KL ADD FRAME FLY

0170 KL DEL FRAME FLY
0176 KL NEW FAST TICKER
017D KL ADD FAST TICKER
0183 KL DEL FAST TICKER
0189 traiter Ticker Chain
01B3 KL ADD TICKER
01C5 KL DEL TICKER
01D2 KL INIT EVENT
01E2 KL EVENT
0219 KL DO SYNC
0227 KL SYNC RESET
022E ajouter sync event
0255 KL NEXT SYNC
0276 KL DONE SYNC
0284 KL DEL SYNCHRONOUS
028D KL DISARM EVENT
0294 KL EVENT DISABLE
029A KL EVENT ENABLE
02A0 KL LOG EXT
02B1 KL FIND COMMAND
0326 KL ROM WALK
0330 KL INIT BACK
0379 Add Event
0388 Delete Event
0397 KL FIXER CONFIGURATION RAM
03C7 KL POLL SYNCHRONOUS
03E7 RST 7 INTERRUPT ENTRY CONT'D
041E KL EXT INTERRUPT ENTRY
042A KL LOW PCHL CONT'D
0430 RST 1 LOW JUMP CONT'D
045F KL FAR PCHL CONT'D
0467 KL FAR ICALL CONT'D
046D RST 3 LOW FAR CALL CONT'D
04BD KL SIDE PCHL CONT'D
04C3 RST 2 LOW SIDE CALL CONT'D
04DB RST 5 FIRM JUMP CONT'D
04F7 KL L ROM ENABLE CONT'D
04FE KL L ROM DISABLE CONT'D
0505 KL U ROM ENABLE CONT'D

050C KL U ROM DISABLE CONT'D
0516 KL ROM RESTORE CONT'D
051F KL ROM SELECT CONT'D
0524 KL PROBE ROM CONT'D
052D KL ROM DESELECT CONT'D
0543 KL CURR SELECTION CONT'D
0547 KL LDIR CONT'D
054D KL LDDR CONT'D
0553 KL ROM OFF & CONFIG. SAVE
056C RST 4 RAM LAM CONT'D
057D KL RAM LAM (IX)

MACHINE PACK

0591 Reset Cont'd
05C5 table 60Hz
05D5 table 50Hz
05ED MC BOOT PROGRAM
061C MC START PROGRAM
0677 démarrage à froid
0688 message initial
06FC sortir message
0705 message erreur de chargement
0738 noms de firme
0776 MC SET MODE
0786 MC CLEAR INKS
078C MC SET INKS
07AA sortir couleur
07B4 MC WAIT FLYBACK
07C0 MC SCREEN OFFSET
07E0 MC RESET PRINTER
07F7 convertir accents
080C MC AFFECTATION DE CARACTERES
081B MC PRINT CHAR
0835 MC WAIT PRINTER
0844 MC SEND PRINTER
0858 MC BUSY PRINTER
0863 MC SOUND REGISTER
0883 Scan Keyboard

JUMP RESTORE

08BD JUMP RESTORE
08DE Main Jump Adress
0A72 BASIC Jump Adr.
0AB4 Move (hl+3) vers ((hl+1)),cnt=(hl)

SCREEN PACK

0ABF SCR INITIALISE
0AD0 SCR RESET
0AE9 SCR SET MODE
0B0C SCR GET MODE
0B17 SCR CLEAR
0B37 SCR SET OFFSET
0B3C SCR SET BASE
0B45 SCR MODIFIER DEBUT ECRAN
0B56 SCR GET LOCATION
0B5D SCR CHAR LIMITS
0B6A SCR CHAR POSTION
0BAF SCR DOT POSITION
0C05 SCR NEXT BYTE
0C11 SCR PREV BYTE
0C1F SCR NEXT LINE
0C39 SCR PREV LINE
0C55 SCR ACCESS
0C71 SCR WRITE
0C74 SCR PIXELS
0C7A XOR Mode
0C7F AND Mode
0C85 OR Mode
0C8A SCR READ
0C8E SCR INK ENCODE
0CA7 SCR INK DECODE
0CD8 Reset couleurs
0CEA SCR SET FLASHING
0CEE SCR GET FLASHING
0CF2 SCR SET INK
0CF7 SCR SET BORDER

0CF8 Set Colour
0D10 aller chercher entrée matrice couleur
0D1A SCR GET INK
0D1F SCR GET BORDER
0D20 Get Colour
0D35 aller chercher adresse ink
0D61 Set Inks on Frame Fly
0D73 Flash Inks
0D87 aller chercher paramètres du jeu de couleurs actuel
0D99 matrice couleurs
0DB9 SCR FILL BOX
0DBD SCR FLOOD BOX
0DE5 SCR CHAR INVERT
0DF8 adresser mémoire couleurs
0E00 SCR HW ROLL
0E44 SCR SW ROLL
0EF9 SCR UNPACK
0F2A SCR REPACK
0F93 SCR HORIZONTAL
0F9B SCR VERTICAL
1052 couleurs défaut

TEXT SCREEN

1074 TXT INITIALISE
1084 TXT RESET
109F Reset Params (toutes les fenêtres)
10E4 TXT STR SELECT
1103 TXT SWAP STREAMS
111E ldir cnt=15
1126 Adr. paramètres fenêtre vers de
1139 fixer paramètres défaut
115A TXT SET COLUMN
1165 TXT SET ROW
1170 TXT SET CURSOR
117C TXT GET CURSOR
1186 fenêtre actuelle haut, gauche + hl
1193 fenêtre actuelle haut, gauche - hl
11A4 Move Cursor

11CA	TXT VALIDATE
11D6	hl à l'intérieur limites fenêtre
1208	TXT WIN ENABLE
1252	TXT GET WINDOW
125F	TXT DRAW/UNDRAW CURSOR
1265	TXT PLACE/REMOVE CURSOR
1276	TXT CUR ON
127E	TXT CUR OFF
1286	TXT CUR ENABLE
1288	Cur Enable Cont'd
1297	TXT CUR DISABLE
1299	Cur Disable Cont'd
12A6	TXT SET PEN
12AB	TXT SET PAPER
12BA	TXT GET PEN
12C0	TXT GET PAPER
12C6	TXT INVERSE
12D4	TXT GET MATRIX
12F2	TXT SET MATRIX
12FE	TXT SET M TABLE
132B	TXT GET M TABLE
1335	TXT WR CHAR
134B	TXT WRITE CHAR
137B	TXT SET BACK
1388	TXT GET BACK
13A8	TXT SET GRAPHIC
13AC	TXT RD CHAR
13BE	TXT UNWRITE CHAR
13FE	TXT OUTPUT
140A	TXT OUT ACTION
1452	TXT VDU DISABLE
1459	TXT VDU ENABLE
1460	FLAG CURSEUR ACTUEL VERS ACCU
1464	copier sauts caractères de commande défaut
1474	sauts caractères de commande défaut
14D4	TXT GET CONTROLS
14E1	bip-bip
14EC	mode transparent activé/désactivé
14F1	instruction INK

14FA instruction BORDER
1501 définir fenêtre
150D instruction SYMBOL
1519 CRSR Left
151E CRSR Right
1523 CRSR Down
1528 CRSR Up
1539 CRSR Home
153F CRSR sur début de ligne
1547 instruction LOCATE
154F TXT CLEAR WINDOW
155E supprimer caractère dans position CRSR
1565 supprimer fenêtre à partir de position CRSR
1578 supprimer fenêtre jusqu'à position CRSR
158F supprimer ligne à partir de position CRSR
1599 supprimer ligne jusqu'à position CRSR

GRAPHICS SCREEN

15A8 GRA INITIALISE
15D7 GRA RESET
15EC NN
15FB GRA MOVE RELATIVE
15FE GRA MOVE ABSOLUTE
1606 GRA ASK CURSOR
160E GRA SET ORIGIN
161C GRA GET ORIGIN
1624 aller chercher position de départ physique
1627 aller chercher position objet physique + fixer curseur
162A GRA CONVERTIR COORD.
165D ajouter coord. act. + coord. rel..
16A5 GRA WIN WIDTH
16EA GRA WIN HEIGHT
1717 GRA GET W WIDTH
172D GRA GET W HEIGHT
1736 GRA CLEAR WINDOW
1767 GRA SET PEN
176E GRA SET PAPER
1775 GRA GET PEN

177A GRA GET PAPER
1780 GRA PLOT RELATIVE
1783 GRA PLOT ABSOLUTE
1786 GRA PLOT
1794 GRA TEST RELATIVE
1797 GRA TEST ABSOLUTE
179A GRA TEST
17A6 GRA LINE RELATIVE
17A9 GRA LINE ABSOLUTE
17AC GRA SAUVER PARAMETRES MASQUE
17B0 GRA SAUVER PARAMETRES MASQUE
17B4 GRA LINE
1940 GRA WR CHAR
19D5 GRA SAUVER PARAMETRES
19D9 GRA FILL

KEYBOARD MANAGER

1B5C KM INITIALISE
1B98 KM RESET
1BBF KM WAIT CHAR
1BC5 KM READ CHAR
1BFA KM CHAR RETURN
1C04 KM EXP BUFFER
1C0A Exp Buffer Cont'd
1C3C Default Exp String
1C46 KM SET EXPAND
1C6A vider buffer d'extension
1CA7 place pour une nouvelle chaîne d'extension?
1CB3 KM GET EXPAND
1CC3 adresse Exp String vers de
1CDB KM WAIT KEY
1CE1 KM READ KEY
1D38 KM GET STATE
1D3C Set State
1D40 KM UPDATE KEY STATE MAP
1DB8 KM TEST BREAK
1DE5 KM GET JOYSTICK
1DF2 KM GET DELAY

1DF6 KM SET DELAY
1DFA KM ARM BREAK
1E0B KM DISARM BREAK
1E19 KM BREAK EVENT
1E2F KM GET REPEAT
1E34 KM SET REPEAT
1E45 KM TEST KEY
1E55 aller chercher bit correspondant à la touche
1E6D masques bits
1EC4 KM GET TRANSLATE
1EC9 KM GET SHIFT
1ECE KM GET CONTROL
1ED1 Get Key Table
1ED8 KM SET TRANSLATE
1EDD KM SET SHIFT
1EE2 KM SET CONTROL
1EE5 Set Key Table
1EEF Key Translation Table
1F3F Key SHIFT Table
1F8F Key CTRL Table

SOUND MANAGER

1FE9 SOUND RESET
2050 SOUND HOLD
206B SOUND CONTINUE
208B Sound Event
20D7 Scan Sound Queues
2114 SOUND QUEUE
21AC SOUND RELEASE
21CE SOUND CHECK
21EB SOUND ARM EVENT
23DB fixer volume
2495 SOUND AMPL ENVELOPE
249A SOUND TONE ENVELOPE
249D copier courbe d'enveloppe
24A6 SOUND A ADRESS
24AB SOUND T ADRESS
24AE aller chercher adresse courbe d'enveloppe

CASSETTE MANAGER

24BC CAS INITIALISE
24CE CAS SET SPEED
24E1 CAS NOISY
24E5 CAS IN OPEN
24FE CAS OUT OPEN
2502 Cass. Open
2550 CAS IN CLOSE
2557 CAS IN ABANDON
257F CAS OUT CLOSE
2599 CAS OUT ABANDON
25A0 CAS IN CHAR
25C6 CAS OUT CHAR
25F6 Check Input Buffer Status
25F9 Check Buffer Status
2603 CAS TEST EOF
2607 CAS RETURN
2618 CAS IN DIRECT
2653 CAS OUT DIRECT
2692 CAS CATALOG
26AC lire File Header
2891 sortir message CAS (# in b)
28F0 sortir message CAS (1 caractère)
2935 messages cassette
29A6 CAS READ
29AF CAS WRITE
29C1 CAS CHECK
29E3 moteur activé & ouvrir clavier
2B3D Cass. Input RD DATA & Test ESC
2BA7 Cass. Output WR DATA
2BBB CAS START MOTOR
2BBF CAS STOP MOTOR
2BC1 CAS RESTORE MOTOR

SCREEN EDITOR

2C02 EDIT
2C42 EDIT exécuter saut

2C72 EDIT table de saut 1
2CAE EDIT table de saut 2
2CBD CRSR UP
2CC1 CRSR DWN
2CC5 CRSR RGHT
2CC9 CRSR LEFT
2CD0 ESC
2CEA message BREAK
2CF1 ENTER
2CFE BIP-BIP
2D02 CRSR RGHT (buffer)
2D0A CRSR DWN (buffer)
2D14 CTRL & CRSR RGHT
2D1D CTRL & CRSR DWN
2D34 CRSR LEFT (buffer)
2D3C CRSR UP (buffer)
2D45 CTRL & CRSR LEFT
2D4F CTRL & CRSR UP
2D81 CTRL & TAB (Flip Insert)
2D8A ajouter caractère
2DC3 DEL
2DCD CLR
2E17 SHFT & CRSR RGHT
2E1C SHFT & CRSR LEFT
2E21 SHFT & CRSR UP
2E26 SHFT & CRSR DWN
2E65 COPY
2F56 caractère de clavier

ARITHMETIQUE

2F73 FLO PI
2F91 FLO COPIER VARIABLE DE (DE) VERS (HL)
2F9F FLO ENTIER VERS VIRGULE FLOTTANTE
2FC8 FLO VALEUR 4 OCTETS VERS VIRGULE FLOTTANTE
2FD1 FLO VALEUR 4 OCTETS PAR 256 VERS ENTIER
2FD9 FLO VIRGULE FLOTTANTE VERS ENTIER
3001 FLO VIRGULE FLOTTANTE VERS ENTIER
3014 FLO FIX

3055 FLO INT
305F FLO
30C6 FLO MULTIPLIER NOMBRE PAR 10^A
3136 FLO RND INIT
3143 FLO SET RANDOM SEED
3159 FLO RND
3188 FLO ALLER CHERCHER DERNIERE VALEUR RND
31B1 FLO LOG10
31B6 FLO LOG
322F FLO EXP
32AC FLO SQR
32AF FLO ELEVATION A LA PUISSANCE
3345 FLO DEG/RAD
3349 FLO COS
3353 FLO SIN
33C8 FLO TAN
33D8 FLO ATN
349E FLO SOUSTRACTION
34A2 FLO ADDITION
3577 FLO MULITIPLICATION
3604 FLO DIVISION
36DF FLO COMPARAISON
3727 FLO SGN
3731 FLO CHANGEMENT DE SIGNE

CHARACTERS

3800-3FFF CHARACTERS

4.2 Références à la RAM système

Voici maintenant pour toutes les adresses utilisées par le système d'exploitation des références croisées aux endroits où elles apparaissent. Ces références peuvent vous être très utiles lorsque vous voudrez manipuler le contenu des adresses de la RAM avec vos propres programmes. Si vous vous apercevez soudain qu'une autre valeur y figure que celle que vous aviez prévue, ces références croisées vous aideront à comprendre pourquoi.

Ici également, nous nous en tenons aux données concernant le CPC 6128.

B100: 0638
B101: 063B
B114: 2DA5 2DBB 2DDE 2DEA
B115: 2C24 2D81 2D85 2D8D
B116: 2DF3 2DFA 2E13 2E41 2EC1
B117: 2DF6
B118: 24E1 2807 28D2
B119: 280C 290F
B11A: 24E5 2550 2557 25F6 2692 26E0 271B 292F
B11B: 263C 269C 26EF
B11D: 25BC 25C1 260F 2613 26F2
B11F: 2743 274E 2760
B12F: 26FC
B130: 26AC
B131: 24FA
B132: 25AA 25B5 25B9 2608 260C 2629 263F 270C
B134: 24F2 261F 2626 26DD
B136: 2706
B137: 24F6
B15F: 24FE 257F 2599 25CA 2656 27D9
B160: 266E 2685 279E
B162: 25EA 25EF 27A1
B164: 2790 27A8
B174: 27CD

B175: 258B 27BF
B176: 2663
B177: 25D4 25E3 25E7 2671 267E 27B6 27CA
B179: 27A4
B17B: 2796 27D2
B17C: 2666
B17E: 266A
B1A4: 26BB 274B 2763 2804
B1B5: 2700
B1B7: 26D9 2709
B1B9: 2022 2072 2094 20BE 2122 214D 21B9
B1BB: 273D
B1BC: 21D1
B1BE: 20E9 2637
B1D5: 21EF
B1E4: 2564 27E5
B1E5: 29E3 2ACD 2AE3
B1E6: 2AC6 2B23
B1E7: 24DC
B1E8: 2B78 2B8B
B1E9: 24D9
B1EA: 2B7C
B1EB: 2B00 2B12 2B16
B1ED: 1FE9 206B
B1EE: 2050 208D 20B7 20D7 2258 2286
B1F0: 201D 20D1 210C 2147 21B4
B1F8: 2000 2296
B237: 229E 22C0
B276: 22A6 22B8
B2A6: 2303 2495 24A6
B2B5: 1FFD 23EF
B396: 249A 24AB
B590: 1B9E
B5D6: 1B6E
B628: 1BCF 1BF0
B629: 1C38
B62A: 1BC6 1BFA
B62B: 1C17 1CC9
B62D: 1C13

B62F: 1C35 1C96 1CA1 1CA7
B630: 1CAC
B631: 1B68 1D12 1D2B 1D38 1D3C
B632: 1CFB
B633: 1D9E 1DF2 1DF6
B634: 1DD8
B635: 1B8A 1D57 1D86 1E4D
B637: 1D4F 1E46
B63B: 1DE5
B63D: 1DB8
B63E: 1DEB
B63F: 1B8D 1D43
B649: 1D40 1D54
B64B: 1D49
B653: 1D7A 1D92 1DA1
B654: 1D7F 1DAC
B655: 1B63
B656: 1E0D 1E19
B657: 1DFD
B686: 1E76 1E86 1EAE
B688: 1E97 1E9D
B68A: 1D96 1E93 1EAA
B68B: 1EC4 1ED8
B68D: 1EC9 1EDD
B68F: 1ECE 1EE2
B691: 1D8B 1E2F 1E37
B692: 1B71
B693: 160E 161C 1640
B695: 1612 1620 1655
B697: 15FE 1606 165E
B699: 1602 160A 1664
B69B: 166A 16C9 1717 1753 1910
B69D: 1673 16CD 171B 1906
B69F: 1680 16FB 172D 174A 18B9 1AE8
B6A1: 1689 16FF 1731 1746 18C3 1B18
B6A3: 0FA5 0FAE 0FB1 0FFF 101C 176A 1775 178D 19C4 1B34
B6A4: 0FF3 1027 175D 1771 177A 19CE
B6A5: 17BD 188F 18C8 18DA 18E6 18EF 18FA 18FF 19D9 1A19 1A44 1AAC 1AC1

B6A7: 17CC 1893 18A2 18AD 18B2 1915 1928 1934 19DF 1A25 1A2C 1A9F 1AA9

B6A9: 1802 1861 19FE 1A4B 1AC6

B6AA: 19E6 1B3A

B6AB: 17EC 1846 1A0B 1A21 1ABD 1AD7 1ADB 1ADF

B6AC: 1A50 1A79

B6AD: 17C4 17E8 1812 181B 18D2 18DD

B6AE: 17D3 17E2 191F 1A5D 1A66 1A94

B6AF: 17DF 1828 1898

B6B0: 17F9 1868 1876 1880 1A76 1A97

B6B2: 17B0 17F2 1820

B6B3: 0FA9 0FB4 0FBA 1012 104C 17AC

B6B4: 0FF7 1021 19C9 19D5

B6B5: 10AF 10B3 10E6 1103 110C

B6B6: 10A1

B726: 10A4 1135 115F 116A 1176 117C 11A7 11AD 1340 1555 156F 1582

B728: 123A 1259

B729: 1166 1186 1193 11EF 1229 1252 1539 1552 1568 157B

B72A: 115B 118C 119B 11DD 11E2 1542 159E

B72B: 11F7 122C 1255 1558 156B

B72C: 11D6 11EA 157E 1593

B72D: 1182 11B2

B72E: 113C 125F 128E 129F 1336 143B 1460

B72F: 10CA 10DA 126B 12A6 12BA 12C9 12CF 1392 13A0 13DB

B730: 11BD 12AB 12C0 13BE 1589

B731: 1377 1384 1388

B733: 13A8 140B

B734: 1321 132B

B735: 1078

B736: 1326 1331

B738: 134F 13C1 13E7

B758: 1413 144E 1465

B759: 142C 1446

B763: 146B

B7C3: 0B0C 0B31

B7C4: 0B3C 0B51 0B56 0B8A 0E2A 0E3D

B7C5: 0B20

B7C6: 0AC7 0B37 0B47 0B59 0B93 0BED 0E32

B7C7: 0C6A 0C71

B7C8: 0C6D
B7D2: 0CEA 0CEE 0D95
B7D3: 0D8E
B7D4: 0CDB 0D92
B7E5: 0D38 0D87
B7F6: 0CE4 0D7C 0D8A
B7F7: 0D0C 0D83
B7F8: 0D61 0D73
B7F9: 0D42 0D55
B802: 0FA1 0FBD
B804: 07E3 0812
B82D: 0066 00F2 011D 0127
B82E: 00EC
B82F: 00F5 00FE 0102
B831: 00E2 00F8 0114 0132 0142 03FE
B832: 010A 014E
B8B4: 009E 00AC 00B1 010E
B8B6: 009A 00A8
B8B8: 00A5
B8B9: 00BF 016A 0170
B8BB: 00C7 017D 0183
B8BD: 00DC 0189 01BF 01C5
B8BF: 00D2 03D0
B8C0: 0256 026E 0287 03D6
B8C1: 022A 03C7
B8C2: 0263 026B 0276 0294 029A 03E0
B8C3: 0230 02B1 0307
B8D3: 02A1 02A5 02BE
B8D5: 0399
B8D6: 0080 0351 0484 04B5 0539 0543
B8D7: 0060 0086
B8D9: 005D 0083 0330 04D5
B8DA: 034E
D3: 0D8E
B7D4: 0CDB 0D92
B7E5: 0D38 0D87
B7F6: 0CE4 0D7C 0D8A
B7F7: 0D0C 0D83
B7F8: 0D61 0D73

B7F9: 0D42 0D55
B802: 0FA1 0FBD
B804: 07E3 0812
B82D: 0066 00F2 011D 0127
B82E: 00EC
B82F: 00F5 00FE 0102
B831: 00E2 00F8 0114 0132 0142 03FE
B832: 010A 014E
B8B4: 009E 00AC 00B1 010E
B8B6: 009A 00A8
B8B8: 00A5
B8B9: 00BF 016A 0170
B8BB: 00C7 017D 0183
B8BD: 00DC 0189 01BF 01C5
B8BF: 00D2 03D0
B8C0: 0256 026E 0287 03D6
B8C1: 022A 03C7
B8C2: 0263 026B 0276 0294 029A 03E0
B8C3: 0230 02B1 0307
B8D3: 02A1 02A5 02BE
B8D5: 0399
B8D6: 0080 0351 0484 04B5 0539 0543
B8D7: 0060 0086
B8D9: 005D 0083 0330 04D5
B8DA: 034E

00 Fin de ligne
01 ':', fin de l'instruction
02 variable entière '%'
03 variable chaîne '$'
04 variable réelle '!'
0D variable sans marque
0E constante 0
0F constante 1
10 constante 2
11 constante 3
12 constante 4
13 constante 5
14 constante 6
15 constante 7
16 constante 8
17 constante 9
19 valeur sur un octet
1A valeur deux octets, décimal
1B valeur deux octets, binaire
1C valeur deux octets, hexa
1D adresse de ligne
1E numéro de ligne
1F valeur à virgule flottante

80	AFTER
81	AUTO
82	BORDER
83	CALL
84	CAT
85	CHAIN
86	CLEAR
87	CLG
88	CLOSEIN
89	CLOSEOUT
8A	CLS
8B	CONT
8C	DATA
8D	DEF
8E	DEFINT
8F	DEFREAL
90	DEFSTR
91	DEG
92	DELETE
93	DIM
94	DRAW
95	DRAWR
96	EDIT
97	ELSE
98	END
99	ENT
9A	ENV
9B	ERASE
9C	ERROR
9D	EVERY
9E	FOR
9F	GOSUB
A0	GOTO
A1	IF
A2	INK
A3	INPUT
A4	KEY
A5	LET
A6	LINE
A7	LIST
A8	LOAD
A9	LOCATE
AA	MEMORY
AB	MERGE
AC	MID$
AD	MODE
AE	MOVE
AF	MOVER
B0	NEXT
B1	NEW
B2	ON
B3	ON BREAK
B4	ON ERROR GOTO 0
B5	ON SQ
B6	OPENIN
B7	OPENOUT
B8	ORIGIN
B9	OUT
BA	PAPER
BB	PEN
BC	PLOT

BD	PLOTR
BE	POKE
BF	PRINT
C0	'
C1	RAD
C2	RANDOMIZE
C3	READ
C4	RELEASE
C5	REM
C6	RENUM
C7	RESTORE
C8	RESUME
C9	RETURN
CA	RUN
CB	SAVE
CC	SOUND
CD	SPEED
CE	STOP
CF	SYMBOL
D0	TAG
D1	TAGOFF
D2	TRON
D3	TROFF
D4	WAIT
D5	WEND
D6	WHILE
D7	WIDTH
D8	WINDOW
D9	ZONE
DA	WRITE
DB	DI
DC	EI
DD	FILL
DE	GRAPHICS
DF	MASK
E0	FRAME
E1	CURSOR
E3	ERL
E4	FN
E5	SPC
E6	STEP
E7	SWAP
EA	TAB
EB	THEN
EC	TO
ED	USING
EE	>
EF	=
F0	>=
F1	<
F2	<>
F3	<=
F4	+
F5	-
F6	*
F7	/
F8	^
F9	'Backslash'
FA	AND
FB	MOD
FC	OR
FD	XOR
FE	NOT
FF	Funktion

Le token &FF précède une fonction. Il peut être suivi des tokens suivants:

00	ABS	71	BIN$
01	ASC	72	DEC$
02	ATN	73	HEX$
03	CHR$	74	INSTR
04	CINT	75	LEFT$
05	COS	76	MAX
06	CREAL	77	MIN
07	EXP	78	POS
08	FIX	79	RIGHT$
09	FRE	7A	ROUND
0A	INKEY	7B	STRING$
0B	INP	7C	TEST
0C	INT	7D	TESTR
0D	JOY	7E	COPYCHR$
0E	LEN	7F	VPOS
0F	LOG		
10	LOG10		
11	LOWER$		
12	PEEK		
13	REMAIN		
14	SGN		
15	SIN		
16	SPACE$		
17	SQ		
18	SQR		
19	STR$		
1A	TAN		
1B	UNT		
1C	UPPER$		
1D	VAL		
40	EOF		
41	ERR		
42	HIMEM		
43	INKEY$		
44	PI		
45	RND		
46	TIME		
47	XPOS		
48	YPOS		
49	DERR		

Le moniteur

Certainement qu'un bon nombre d'entre vous brûlent de découvrir ce que renferme précisément le listing de la ROM, qui n'est rien d'autre que le contenu symbolique du système d'exploitation. Malheureusement, un peu de persévérance vous sera nécessaire. Si vous ne disposez pas déjà d'un moniteur de langage machine, il faut d'abord que vous tapiez celui que nous publions ici.

Excepté deux petites routines machine, l'une pour lire un octet dans la mémoire, l'autre pour aller chercher un octet dans un fichier, le programme est entièrement écrit en Basic. Comme toutefois le jeu d'instructions tout entier est d'abord placé dans des tableaux, le désassembleur reste cependant très rapide.

Nous devons cependant confesser une insuffisance. Le procédé utilisé ne permet pas de traiter certaines instructions du type (IX+xx). Si une telle instruction apparaît, le message "!! instruction spéciale ...." apparaîtra dans le listing. En cas de besoin, il faudra donc que vous insériez vous-même cette instruction, en utilisant sa forme binaire. De telles instructions sont cependant vraiment rares. Elles apparaissent seulement deux ou trois fois dans le Sound Manager.

Par ailleurs, la représentation des instructions ne correspond pas tout à fait au standard du Z80. C'est ainsi par exemple que, dans notre moniteur, les valeurs immédiates sont marquées par une dièze les précédant. Les valeurs de deux octets non marquées de cette façon sont des adresses.

Vous avez la possibilité de désassembler la RAM, la ROM ou un fichier. Cette dernière possibilité n'est que rarement offerte par d'autres programmes. Il est intéressant de l'utiliser lorsque le programme à traiter ne peut tenir en mémoire en même temps qu'un programme Basic.

Avant que nous n'en venions à la description des instructions, encore un petit conseil: laissez tout d'abord de côté les lignes 20 à 40, de façon à ce qu'une erreur de syntaxe provoquée par des fautes de frappe ne soit pas inhibée. De toute façon, si vous n'avez pas l'intention de travailler à partir d'un fichier, ces lignes peuvent être négligées car elles servent uniquement à empêcher le "nettoyage" de la mémoire qui est sinon inévitable lors de l'ouverture d'un fichier. Vous devez également déduire de ces lignes que vous devez appeler le programme "mimo.bas", pour que OPENIN trouve bien un fichier.

Venons-en maintenant aux quelques instructions disponibles. Le principe est que tous les paramètres doivent être placés immédiatement à la suite de l'instruction, en hexadécimal. Si vous voulez, par exemple, fixer l'adresse actuelle sur $0048, entrez: m48>ENTER<

d désassembler à partir de l'adresse actuelle. Cette fonction est interrompue par la frappe d'une touche quelconque.

f f doit être immédiatement suivi du nom de fichier complet du fichier que vous voulez traiter. Avec l'entrée suivante, vous donnez l'adresse relative avec laquelle le fichier doit apparaître à l'écran. Cela ne sert qu'à la présentation. Le fichier lui-même commence de toute façon à partir du début. Les instructions d'affichage ultérieures se rapportent alors à ce fichier. Le mode fichier est interrompu par la fonction m.

i écrire des octets dans la mémoire. Cette instruction ne nécessite aucun paramètre. Les octets sont réclamés un par un, à partir de l'adresse actuelle. Cette fonction se termine lorsque vous effectuez une entrée vide.

o fixer le fichier de sortie. 0 est le cas normal qui amène l'affichage entier en mode 1 sur l'écran. 1 amène l'affichage en mode 2 à l'écran, divisé de façon à ce que le tiers supérieur soit responsable de l'affichage normal

de la mémoire alors que le reste est réservé au désassembleur. Lorsque vous passez de l'affichage au désassembleur et vice versa, les fenêtres sont conservées. Enfin, 8 dirige la sortie sur l'imprimante.

m fixe l'adresse actuelle à laquelle se réfèrent toutes les instructions ultérieures.

b fixe la configuration mémoire. L'octet réclamé a la structure qui a été décrite plus haut dans cet ouvrage. FE sélectionne par exemple les deux ROMs intégrées plus la RAM placée entre ces deux ROMs, FF ne sélectionne que la RAM.

$ convertit un paramètre décimal en hexadécimal.

% convertit un paramètre hexadécimal (de quatre chiffres maximum) en un nombre décimal.

x met fin au programme et restaure la limite de la mémoire.

? effectue un warmstart et affiche la liste des instructions.

>ENTER< entré seul liste le contenu de la mémoire en hexadécimal et en ASCII.

Il nous reste encore à espérer que la frappe du listing suivant ne vous posera pas trop de problèmes. Notez que '^' dans le listing correspond à la flèche verticale.

```
10 top=HIMEM
20 ON ERROR GOTO 40
30 OPENIN "mimo.bas"
40 RESUME NEXT
50 MEMORY HIMEM-1
60 CLOSEIN
70 him=HIMEM-256
80 ZONE 8:lf=0
90 mpb=him-20:MEMORY mpb-1
100 GOSUB 1350:ms=&FE
110 CLS:INK 3,6:b0=1:b1=24:b2=22:b3=0
120 DIM l%(4,255),mn$(4,255),pu%(15)
130 GOSUB 1010:a=0
140 bs$=STRING$(32,8):bl$=SPACE$(30)
150 IF plf=1 THEN lf=0:plf=0
160 MODE 1:PRINT:PRINT: PRINT"c =     programme machine"
170 PRINT"d =    Désassembler"
180 PRINT"f = Fichier"
190 PRINT"i = Insérer octets"
200 PRINT"o = Output-lf#"
210 PRINT"m = Adresse mémoire"
220 PRINT"b = Sélection de banque"
230 PRINT"$ = Decimal - > Hex"
240 PRINT"% = Hex - > Decimal"
250 PRINT"x = Fin"
260 PRINT"? = Warmstart"
270 PRINT:GOTO 290
280 IF lf=0 OR lf>7 THEN MODE 1
290 BORDER b0:INK 0,b0:INK 1,b1:PRINT:PRINT"bank= ";HEX$(ms,2):PRINT "mem = ";
    HEX$(a,4):i=a:PRINT"lf# =";lf:PRINT
300 INPUT">",h$:hl$=LEFT$(h$,1)
310 IF h$="?" THEN GOTO 150
320 IF h$="x" THEN MEMORY top:MODE 1:END
330 IF hl$<>"o" THEN 370
340 lf=VAL(RIGHT$(h$,1)):IF lf=0 OR lf>7 THEN plf=0:GOTO 280
350 IF plf=0 THEN MODE 2:WINDOW #0,1,80,25,25:WINDOW #1,1,80,1,8:
    WINDOW #2,1,80,9,25:plf=1
360 GOTO 290
```

```
370 IF hl$="$" THEN PRINT HEX$(VAL(RIGHT$(h$,LEN(h$)-1))):GOTO 290
380 IF hl$<>"%" THEN 410
390 xx=(VAL("&"+RIGHT$(h$,LEN(h$)-1))):IF xx<0 THEN xx=xx+65536
400 PRINT xx:GOTO 290
410 IF hl$<>"m" THEN 460
420 IF file=1 THEN file=0:CLOSEIN
430 IF LEN(h$)=1 THEN 280
440 a=VAL("&"+RIGHT$(h$,LEN(h$)-1)):IF a<0 THEN a=a+65536
450 padp=a-1:GOTO 280
460 IF hl$<>"b" THEN 490
470 re=VAL("&"+RIGHT$(h$,LEN(h$)-1)):IF re>255 OR re<0 THEN
    PRINT"Il faut une valeur hexa sur 2 octets":GOTO 280
480 ms=re:GOTO 280
490 IF hl$<>"f" THEN 570
500 IF file=1 THEN CLOSEIN
510 ON ERROR GOTO 530
520 OPENIN MID$(h$,2)
530 RESUME NEXT
540 INPUT"base  (hex) ";h$
550 h$="m"+h$
560 file=1:GOTO 440
570 REM
580 IF hl$="d" THEN i=a:GOTO 810
590 IF hl$="c" THEN CALL a:GOTO 280
600 IF hl$="i" THEN 780
610 IF LEN(h$)<2 THEN h$="00"
620 bis=VAL("&"+RIGHT$(h$,LEN(h$)-1)):IF bis<1 THEN bis=bis+65536
630 IF plf=0 THEN MODE 2 ELSE lf=1
640 BORDER b2:INK 0,b2:INK 1,b3
650 ON file GOTO 670
660 a=INT(a/16)*16
670 FOR i=a TO bis  STEP 16
680 PRINT#lf,HEX$(i,4);":";:FOR j=0 TO 15
690 pad=i+j:GOSUB 1520:PRINT#lf," ";HEX$(mv,2);
700 NEXT j:PRINT#lf,TAB(60);
710 FOR j=0 TO 15:pad=i+j:GOSUB 1520:he=(mv AND 127)
720 IF he<32 OR he=127 THEN he=46
730 PRINT#lf,CHR$(he);:NEXT j:PRINT#lf
740 IF INKEY$<>"" THEN a=i:i=65535:ELSE a=i+16
```

```
750 NEXT
760 IF lf<>8 THEN INPUT " appuyer <ENTER> quand fini"; re$
770 GOTO 280
780 i=a
790 PRINT HEX$(i,4);": ";:INPUT"",d$:IF d$="" THEN 280
800 POKE i,VAL("&"+d$):i=i+1:GOTO 790
810 IF plf=1 THEN lf=2:PRINT#lf,CHR$(11);
820 IF LEN(h$)=1 THEN h$="00"
830 bis=VAL("&"+RIGHT$(h$,LEN(h$)-1)):IF bis<1 THEN bis=bis+65536
840 pa=a
850 PAPER 0:IF INKEY$ <>"" THEN a=pa:PRINT#lf:GOTO 280
860 IF pa>bis THEN a=pa:PRINT#lf:GOTO 760
870 pad=pa:GOSUB 1520:op=mv:ad=pa:pa=pa+1
880 IF lf=8 THEN PRINT#lf,LEFT$(bl$,10);
890 PRINT#lf,HEX$(ad,4);"  ";:xx=0
900 PRINT#lf,HEX$(op,2);
910 se=0:GOSUB 1700:IF LEFT$(mn$,1)="?" THEN 1070
920 se=xx:GOSUB 1700:IF mn$="" THEN PAPER 3:PRINT#lf,"        ????":
    PAPER 0:GOTO 850
930 ON l%(xx,op) GOTO 980,970,960,950
940 ON l%(xx,op)-1 GOTO 980,970,960,950
950 pad=pa:GOSUB 1520:PRINT#lf,HEX$(mv,2);:pa=pa+1
960 pad=pa:GOSUB 1520:PRINT#lf,HEX$(mv,2);:pa=pa+1
970 pad=pa:GOSUB 1520:PRINT#lf,HEX$(mv,2);:pa=pa+1
980 PRINT#lf,LEFT$(bl$,(4-l%(xx,op))*2+2);
990 GOSUB 1090
1000 GOTO 850
1010 PRINT:PAPER 3:PRINT"veuillez patienter";:PAPER 0:PRINT:FOR I=0 TO 4:
     FOR j=0 TO 255
1020 READ a:l%(i,j)=a
1030 NEXT j,i
1040 FOR i=0 TO 4:FOR j=0 TO 255
1050 READ mn$:mn$(i,j)=mn$
1060 NEXT:NEXT:RETURN
1070 xx=l%(0,op):pad=pa:GOSUB 1520:op=mv:se=xx:GOSUB 1700:IF mn$="" THEN 920
1080 PRINT#lf,HEX$(op,2);:pa=pa+1:GOTO 940
1090 se=xx:GOSUB 1700:ln=LEN(mn$)
1100 IF mn$=pmn$ THEN PAPER 3
1110 pmn$=mn$:ppn=1
```

```
1120 IF MID$(mn$,ln-3,4)="+/-^" THEN mn$=LEFT$(mn$,ln-4):GOTO 1230
1130 pn=INSTR(mn$,"*"):IF pn<>0 THEN PRINT#lf,LEFT$(mn$,pn-1);:GOTO 1170
1140 pn=INSTR(ppn,mn$,"^"):IF pn<>0 THEN PRINT#lf,MID$(mn$,ppn,pn-ppn);:
     GOTO 1220
1150 PRINT#lf,mn$;
1160 PRINT#lf:RETURN
1170 pad=pa-2:GOSUB 1520:ar=mv:pn=pn+1
1180 IF pn>ln THEN xz=ar:PRINT#lf,HEX$(xz,2);:GOTO 1160
1190 ppn=pn:IF MID$(mn$,pn,1)<>"^" THEN xz=ar:PRINT#lf,HEX$(xz,2);:GOTO 1140
1200 pn=pn+1:pad=pa-1:GOSUB 1520:yy=256*mv+ar:PRINT#lf,HEX$(yy,4);
1210 PRINT#lf,MID$(mn$,pn):RETURN
1220 pn=pn+1:pad=pa-1:GOSUB 1520:ar=mv:xz=ar:PRINT#lf,HEX$(xz,2);:GOTO 1210
1230 PRINT#lf,mn$;
1240 pn=pn+1:pad=pa-1:GOSUB 1520:ar=mv:yy=ad+2+ar+(ar>127)*256:
     PRINT#lf,HEX$(yy,4);
1250 PRINT#lf:RETURN
1260 sp=1
1270 WHILE MID$(mn$,sp,1)<>" ": sp=sp+1:WEND
1280 WHILE MID$(mn$,sp,1)=" ": sp=sp+1:WEND
1290 ad=cn+VAL(RIGHT$(mn$,LEN(mn$)-sp+1))
1300 ha=INT(ad/256):la=ad-ha*256
1310 PRINT#lf,"  ($";HEX$(ha,2);HEX$(la,2);")":RETURN
1320 IF MID$(mn$,sp,1)="-" THEN 1340
1330 ad=cn+ar:GOTO 1300
1340 ad=cn+ar-256:GOTO 1300
1350 POKE mpb,&DF
1360 po=mpb+4:ph=INT(po/256):pl=po-ph*256
1370 POKE mpb+1,pl:POKE mpb+2,ph
1380 POKE mpb+3,&C9
1390 po=mpb+7:ph=INT(po/256):pl=po-ph*256
1400 POKE mpb+4,pl:POKE mpb+5,ph
1410 POKE mpb+7,&3A
1420 by=mpb+14:ph=INT(by/256):pl=by-ph*256
1430 POKE mpb+10,&32
1440 POKE mpb+11,pl:POKE mpb+12,ph
1450 POKE mpb+13,&C9
1460 DATA &c1,&d1,&f1,&e1,&f5,&d5,&c5,&cd,&80,&bc,&f5,&d1,&72,&23,&73,&c9
1470 FOR i=1 TO 16
1480 READ a
```

```
1490 mp$=mp$+CHR$(a)
1500 NEXT i
1510 RETURN
1520 IF pad>65535 THEN RETURN
1530 ON file GOTO 1600
1540 IF ms=255 THEN mv=PEEK(pad):RETURN
1550 ph=INT(pad/256):pl=pad-ph*256
1560 POKE mpb+8,pl:POKE mpb+9,ph
1570 POKE mpb+6,ms
1580 CALL mpb
1590 mv=PEEK(by):RETURN
1600 IF padp<pad THEN GOSUB 1630
1610 mv=pu%(pad MOD(16))
1620 RETURN
1630 ret%=0:mpp=@mp$
1640 getf=PEEK(mpp+1)+256*PEEK(mpp+2)
1650 CALL getf,@ret%
1660 mv=ret% AND 255: IF (ret% AND &100)=0 THEN mv=0
1670 padp=padp+1:pu%(padp MOD(16))=mv
1680 IF padp<pad GOTO 1650
1690 RETURN
1700 mn$=mn$(se,op):RETURN
1710 DATA 1 , 3 , 1 , 1 , 1 , 1 , 2 , 1
1720 DATA 1 , 1 , 1 , 1 , 1 , 1 , 2 , 1
1730 DATA 2 , 3 , 1 , 1 , 1 , 1 , 2 , 1
1740 DATA 2 , 1 , 1 , 1 , 1 , 1 , 2 , 1
1750 DATA 2 , 3 , 3 , 1 , 1 , 1 , 2 , 1
1760 DATA 2 , 1 , 3 , 1 , 1 , 1 , 2 , 1
1770 DATA 2 , 3 , 3 , 1 , 1 , 1 , 2 , 1
1780 DATA 2 , 1 , 3 , 1 , 1 , 1 , 2 , 1
1790 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1800 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1810 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1820 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1830 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1840 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1850 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1860 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1870 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
```

```
1880 DATA 0 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1890 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1900 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1910 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1920 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1930 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1940 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1950 DATA 1 , 1 , 3 , 3 , 3 , 1 , 2 , 1
1960 DATA 1 , 1 , 3 , 4 , 3 , 3 , 2 , 1
1970 DATA 1 , 1 , 3 , 2 , 3 , 1 , 2 , 1
1980 DATA 1 , 1 , 3 , 2 , 3 , 2 , 2 , 1
1990 DATA 1 , 1 , 3 , 1 , 3 , 1 , 2 , 1
2000 DATA 1 , 1 , 3 , 1 , 3 , 1 , 2 , 1
2010 DATA 1 , 1 , 3 , 1 , 3 , 1 , 2 , 1
2020 DATA 1 , 1 , 3 , 1 , 3 , 3 , 2 , 1
2030 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2040 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2050 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2060 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2070 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2080 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2090 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2100 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2110 DATA 2 , 2 , 2 , 4 , 2 , 2 , 2 , 2
2120 DATA 2 , 2 , 2 , 4 , 0 , 2 , 0 , 2
2130 DATA 2 , 2 , 2 , 4 , 0 , 0 , 2 , 2
2140 DATA 2 , 2 , 2 , 4 , 0 , 0 , 2 , 2
2150 DATA 2 , 2 , 2 , 4 , 0 , 0 , 0 , 2
2160 DATA 2 , 2 , 2 , 4 , 0 , 0 , 0 , 2
2170 DATA 2 , 0 , 2 , 4 , 0 , 0 , 0 , 0
2180 DATA 2 , 2 , 2 , 4 , 0 , 0 , 0 , 0
2190 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2200 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2210 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2220 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2230 DATA 2 , 2 , 2 , 2 , 0 , 0 , 0 , 0
2240 DATA 2 , 2 , 2 , 2 , 0 , 0 , 0 , 0
2250 DATA 2 , 2 , 2 , 2 , 0 , 0 , 0 , 0
2260 DATA 2 , 2 , 2 , 2 , 0 , 0 , 0 , 0
```

```
2270 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2280 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2290 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2300 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2310 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2320 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2330 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2340 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2350 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2360 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2370 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2380 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2390 DATA 0 , 4 , 4 , 2 , 0 , 0 , 0 , 0
2400 DATA 0 , 2 , 4 , 2 , 0 , 0 , 0 , 0
2410 DATA 0 , 0 , 0 , 0 , 3 , 3 , 4 , 0
2420 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2430 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2440 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2450 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2460 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2470 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2480 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2490 DATA 3 , 3 , 3 , 3 , 3 , 3 , 0 , 3
2500 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2510 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2520 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2530 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2540 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2550 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2560 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2570 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2580 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2590 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2600 DATA 0 , 0 , 0 , 4 , 0 , 0 , 0 , 0
2610 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2620 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2630 DATA 0 , 2 , 0 , 2 , 0 , 2 , 0 , 0
2640 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2650 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
```

```
2660 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2670 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2680 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2690 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2700 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2710 DATA 0 , 4 , 4 , 2 , 0 , 0 , 0 , 0
2720 DATA 0 , 2 , 4 , 2 , 0 , 0 , 0 , 0
2730 DATA 0 , 0 , 0 , 0 , 3 , 3 , 4 , 0
2740 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2750 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2760 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2770 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2780 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2790 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2800 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2810 DATA 3 , 3 , 3 , 3 , 3 , 3 , 0 , 3
2820 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2830 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2840 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2850 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2860 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2870 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2880 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2890 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2900 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2910 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2920 DATA 0 , 0 , 0 , 4 , 0 , 0 , 0 , 0
2930 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2940 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2950 DATA 0 , 2 , 0 , 2 , 0 , 2 , 0 , 0
2960 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2970 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2980 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2990 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3000 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3010 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3020 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3030 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3040 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
```

```
3050 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
3060 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3070 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3080 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3090 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3100 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3110 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3120 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3130 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3140 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3150 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3160 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3170 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3180 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3190 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3200 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3210 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3220 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3230 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3240 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3250 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3260 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3270 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3280 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3290 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3300 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3310 DATA "nop     ","ld      bc,#*^","ld      (bc),a","inc     bc",
     "inc     b","dec     b","ld      b,#^","rlca    "
3320 DATA "ex      af,af'","add     hl,bc","ld      a,(bc)","dec     bc",
     "inc     c","dec     c","ld      c,#^","rrca    "
3330 DATA "djnz    +/-^","ld      de,#*^","ld      (de),a","inc     de",
     "inc     d","dec     d","ld      d,#^","rla     "
3340 DATA "jr      +/-^","add     hl,de","ld      a,(de)","dec     de",
     "inc     e","dec     e","ld      e,#^","rra     "
3350 DATA "jr      nz,+/-^","ld      hl,#*^","ld      *^,hl","inc     hl",
     "inc     h","dec     h","ld      h,#^","daa     "
3360 DATA "jr      z,+/-^","add     hl,hl","ld      hl,*^","dec     hl",
     "inc     l","dec     l","ld      l,#^","cpl     a"
3370 DATA "jr      nc,+/-^","ld      sp,#*^","ld      *^,a","inc     sp",
```

```
     "inc     (hl)","dec     (hl)","ld      (hl),#^","scf     "
3380 DATA "jr      c,+/-^","add     hl,sp","ld      a,*^","dec     sp",
     "inc     a","dec     a","ld      a,#^","ccf     "
3390 DATA "ld      b,b","ld      b,c","ld      b,d","ld      b,e",
     "ld      b,h","ld      b,l","ld      b,(hl)","ld      b,a"
3400 DATA "ld      c,b","ld      c,c","ld      c,d","ld      c,e",
     "ld      c,h","ld      c,l","ld      c,(hl)","ld      c,a"
3410 DATA "ld      d,b","ld      d,c","ld      d,d","ld      d,e",
     "ld      d,h","ld      d,l","ld      d,(hl)","ld      d,a"
3420 DATA "ld      e,b","ld      e,c","ld      e,d","ld      e,e",
     "ld      e,h","ld      e,l","ld      e,(hl)","ld      e,a"
3430 DATA "ld      h,b","ld      h,c","ld      h,d","ld      h,e",
     "ld      h,h","ld      h,l","ld      h,(hl)","ld      h,a"
3440 DATA "ld      l,b","ld      l,c","ld      l,d","ld      l,e",
     "ld      l,h","ld      l,l","ld      l,(hl)","ld      l,a"
3450 DATA "ld      (hl),b","ld      (hl),c","ld      (hl),d","ld      (hl),e",
     "ld      (hl),h","ld      (hl),l","halt    ","ld      (hl),a"
3460 DATA "ld      a,b","ld      a,c","ld      a,d","ld      a,e",
     "ld      a,h","ld      a,l","ld      a,(hl)","ld      a,a"
3470 DATA "add     a,b","add     a,c","add     a,d","add     a,e",
     "add     a,h","add     a,l","add     a,(hl)","add     a,a"
3480 DATA "","adc     a,c","adc     a,d","adc     a,e","adc     a,h",
     "adc     a,l","adc     a,(hl)","adc     a,a"
3490 DATA "sub     a,b","sub     a,c","sub     a,d","sub     a,e",
     "sub     a,h","sub     a,l","sub     a,(hl)","sub     a,a"
3500 DATA "sbc     a,b","sbc     a,c","sbc     a,d","sbc     a,e",
     "sbc     a,h","sbc     a,l","sbc     a,(hl)","sbc     a,a"
3510 DATA "and     a,b","and     a,c","and     a,d","and     a,e",
     "and     a,h","and     a,l","and     a,(hl)","and     a,a"
3520 DATA "xor     a,b","xor     a,c","xor     a,d","xor     a,e",
     "xor     a,h","xor     a,l","xor     a,(hl)","xor     a,a"
3530 DATA "or      a,b","or      a,c","or      a,d","or      a,e",
     "or      a,h","or      a,l","or      a,(hl)","or      a,a"
3540 DATA "cp      a,b","cp      a,c","cp      a,d","cp      a,e",
     "cp      a,h","cp      a,l","cp      a,(hl)","cp      a,a"
3550 DATA "ret     nz","pop     bc","jp      nz,*^","jp      *^",
     "call    nz,*^","push    bc","add     a,#^","rst     0"
3560 DATA "ret     z","ret     ","jp      z,*^","?","call    z,*^",
     "call    *^","adc     a,#^","rst     1"
```

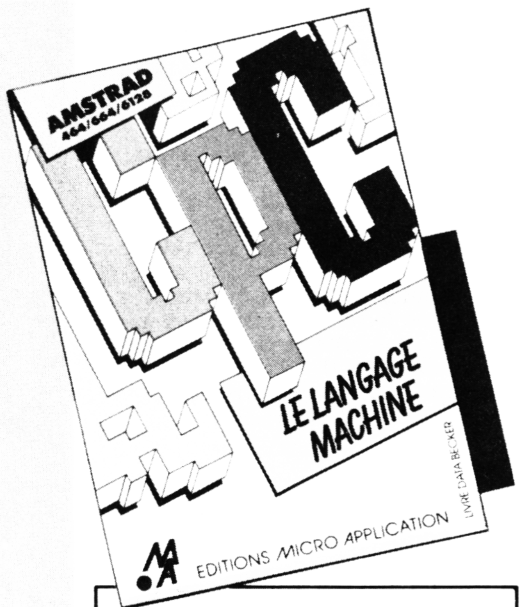
```
3570 DATA "ret     nc","pop     de","jp      nc,*^","out       a",
     "call    nc,*^","push    de","sub     a,#^","rst     2"
3580 DATA "ret     c","exx     ","jp      c,*^","in      a,(^)","call    c,*^",
     "?","sbc     a,#^","rst     3"
3590 DATA "ret     po","pop     hl","jp      po,*^","ex      (sp),hl",
     "call    po,*^","push    hl","and     a,#^","rst     4"
3600 DATA "ret     pe","jp      (hl)","jp      pe,*^","ex      de,hl",
     "call    pe,*^","?","xor     a,#^","rst     5"
3610 DATA "ret     p","pop     af","jp      p,*^","di      ","call    p,*^",
     "push    af","or      a,#^","rst     6"
3620 DATA "ret     m","ld      sp,hl","jp      m,*^","ei      ","call    m,*^",
     "?","cp      a,#^","rst     7"
3630 DATA "","","","","","","",""
3640 DATA "","","","","","","",""
3650 DATA "","","","","","","",""
3660 DATA "","","","","","","",""
3670 DATA "","","","","","","",""
3680 DATA "","","","","","","",""
3690 DATA "","","","","","","",""
3700 DATA "","","","","","","",""
3710 DATA "in      b,(c)","out     (c),b","sbc     hl,bc","ld      *^,bc",
     "neg     a","retn    ","im      0","ld      i,a"
3720 DATA "in      c,(c)","out     (c),c","adc     hl,bc","ld      bc,*^","",
     "reti    ","","ld      r,a"
3730 DATA "in      d,(c)","out     (c),d","sbc     hl,de","ld      *^,de","",
     "","im      1","ld      a,i"
3740 DATA "in      e,(c)","out     (c),e","adc     hl,de","ld      de,*^","",
     "","im      2","ld      a,r"
3750 DATA "in      h,(c)","out     (c),h","sbc     hl,hl","ld      *^,hl","",
     "","","rld     a,(hl)"
3760 DATA "in      l,(c)","out     (c),l","adc     hl,hl","ld      hl,*^","",
     "","","rld     a,(hl)"
3770 DATA "in      f,(c)","","sbc     hl,sp","ld      *^,sp","","","",""
3780 DATA "in      a,(c)","out     (c),a","adc     hl,sp","ld      sp,*^","",
     "","",""
3790 DATA "","","","","","","",""
3800 DATA "","","","","","","",""
3810 DATA "","","","","","","",""
3820 DATA "","","","","","","",""
```

```
3830 DATA "ldi     (de),(hl)","cpi     a,(hl)","ini     (hl),(c)",
     "outi    (c),(hl)","","","",""
3840 DATA "ldd     (de),(hl)","cpd     a,(hl)","ind     (hl),(c)",
     "outd    (c),(hl)","","","",""
3850 DATA "ldir    (de),(hl)","cpir    a,(hl)","inir    (hl),(c)",
     "otir    (c),(hl)","","","",""
3860 DATA "lddr    (de),(hl)","cpdr    a,(hl)","indr    (hl),(c)",
     "otdr    (c),(hl)","","","",""
3870 DATA "","","","","","","",""
3880 DATA "","","","","","","",""
3890 DATA "","","","","","","",""
3900 DATA "","","","","","","",""
3910 DATA "","","","","","","",""
3920 DATA "","","","","","","",""
3930 DATA "","","","","","","",""
3940 DATA "","","","","","","",""
3950 DATA "","","","","","","",""
3960 DATA "","add     ix,bc","","","","","",""
3970 DATA "","","","","","","",""
3980 DATA "","add     ix,de","","","","","",""
3990 DATA "","ld      ix,#*^","ld      *^,ix","inc     ix","","","",""
4000 DATA "","add     ix,ix","ld      ix,*^","dec     ix","","","",""
4010 DATA "","","","","inc     (ix+^)","dec     (ix+^)","ld      (ix+*),#^",""
4020 DATA "","add     ix,sp","","","","","",""
4030 DATA "","","","","","","ld      b,(ix+^)",""
4040 DATA "","","","","","","ld      c,(ix+^)",""
4050 DATA "","","","","","","ld      d,(ix+^)",""
4060 DATA "","","","","","","ld      e,(ix+^)",""
4070 DATA "","","","","","","ld      h,(ix+^)",""
4080 DATA "","","","","","","ld      l,(ix+^)",""
4090 DATA "ld      (ix+^),b","ld      (ix+^),c","ld      (ix+^),d",
     "ld      (ix+^),e","ld      (ix+^),h","ld      (ix+^),l","",
     "ld      (ix+^),a"
4100 DATA "","","","","","","ld      a,(ix+^)",""
4110 DATA "","","","","","","add     a,(ix+^)",""
4120 DATA "","","","","","","adc     a,(ix+^)",""
4130 DATA "","","","","","","sub     a,(ix+^)",""
4140 DATA "","","","","","","",""
4150 DATA "","","","","","","and     a,(ix+^)",""
```

```
4160 DATA "","","","","","","xor      a,(ix+^)",""
4170 DATA "","","","","","","or       a,(ix+^)",""
4180 DATA "","","","","","","cp       a,(ix+^)",""
4190 DATA "","","","","","","",""
4200 DATA "","","","!!!      inst. spéciale avec  (ix+*)","","","",""
4210 DATA "","","","","","","",""
4220 DATA "","","","","","","",""
4230 DATA "","pop      ix","","ex       (sp),ix","","push     ix","",""
4240 DATA "","jmp      (ix)","","","","","",""
4250 DATA "","","","","","","",""
4260 DATA "","ld       sp,ix","","","","","",""
4270 DATA "","","","","","","",""
4280 DATA "","add      iy,bc","","","","","",""
4290 DATA "","","","","","","",""
4300 DATA "","add      iy,de","","","","","",""
4310 DATA "","ld       iy,#*^","ld       *^,iy","inc      iy","","","",""
4320 DATA "","add      iy,iy","ld       iy,*^","dec      iy","","","",""
4330 DATA "","","","","inc      (iy+^)","dec      (iy+^)","ld       (iy+*),#^",""
4340 DATA "","add      iy,sp","","","","","",""
4350 DATA "","","","","","","ld       b,(iy+^)",""
4360 DATA "","","","","","","ld       c,(iy+^)",""
4370 DATA "","","","","","","ld       d,(iy+^)",""
4380 DATA "","","","","","","ld       e,(iy+^)",""
4390 DATA "","","","","","","ld       h,(iy+^)",""
4400 DATA "","","","","","","ld       l,(iy+^)",""
4410 DATA "ld       (iy+^),b","ld       (iy+^),c","ld       (iy+^),d",
     "ld       (iy+^),e","ld       (iy+^),h","ld       (iy+^),l","",
     "ld       (iy+^),a"
4420 DATA "","","","","","","ld       a,(iy+^)",""
4430 DATA "","","","","","","add      a,(iy+^)",""
4440 DATA "","","","","","","adc      a,(iy+^)",""
4450 DATA "","","","","","","sub      a,(iy+^)",""
4460 DATA "","","","","","","sbc      a,(iy+^)",""
4470 DATA "","","","","","","and      a,(iy+^)",""
4480 DATA "","","","","","","xor      a,(iy+^)",""
4490 DATA "","","","","","","or       a,(iy+^)",""
4500 DATA "","","","","","","cp       a,(iy+^)",""
4510 DATA "","","","","","","",""
4520 DATA "","","","!!!      inst. spéciale avec  (iy+*)","","","",""
```

```
4530 DATA "","","","","","","",""
4540 DATA "","","","","","","",""
4550 DATA "","pop     iy","","ex      (sp),iy","","push    iy","",""
4560 DATA "","jmp     (iy)","","","","","",""
4570 DATA "","","","","","","",""
4580 DATA "","ld      sp,iy","","","","","",""
4590 DATA "rlc     b","rlc     c","rlc     d","rlc     e","rlc     h",
     "rlc     l","rlc     (hl)","rlc     a"
4600 DATA "rrc     b","rrc     c","rrc     d","rrc     e","rrc     h",
     "rrc     l","rrc     (hl)","rrc     a"
4610 DATA "rl      b","rl      c","rl      d","rl      e","rl      h",
     "rl      l","rl      (hl)","rl      a"
4620 DATA "rr      b","rr      c","rr      d","rr      e","rr      h",
     "rr      l","rr      (hl)","rr      a"
4630 DATA "sla     b","sla     c","sla     d","sla     e","sla     h",
     "sla     l","sla     (hl)","sla     a"
4640 DATA "sra     b","sra     c","sra     d","sra     e","sra     h",
     "sra     l","sra     (hl)","sra     a"
4650 DATA "","","","","","","",""
4660 DATA "srl     b","srl     c","srl     d","srl     e","srl     h",
     "srl     l","srl     (hl)","srl     a"
4670 DATA "bit     0,b","bit     0,c","bit     0,d","bit     0,e",
     "bit     0,h","bit     0,l","bit     0,(hl)","bit     0,a"
4680 DATA "bit     1,b","bit     1,c","bit     1,d","bit     1,e",
     "bit     1,h","bit     1,l","bit     1,(hl)","bit     1,a"
4690 DATA "bit     2,b","bit     2,c","bit     2,d","bit     2,e",
     "bit     2,h","bit     2,l","bit     2,(hl)","bit     2,a"
4700 DATA "bit     3,b","bit     3,c","bit     3,d","bit     3,e",
     "bit     3,h","bit     3,l","bit     3,(hl)","bit     3,a"
4710 DATA "bit     4,b","bit     4,c","bit     4,d","bit     4,e",
     "bit     4,h","bit     4,l","bit     4,(hl)","bit     4,a"
4720 DATA "bit     5,b","bit     5,c","bit     5,d","bit     5,e",
     "bit     5,h","bit     5,l","bit     5,(hl)","bit     5,a"
4730 DATA "bit     6,b","bit     6,c","bit     6,d","bit     6,e",
     "bit     6,h","bit     6,l","bit     6,(hl)","bit     6,a"
4740 DATA "bit     7,b","bit     7,c","bit     7,d","bit     7,e",
     "bit     7,h","bit     7,l","bit     7,(hl)","bit     7,a"
4750 DATA "res     0,b","res     0,c","res     0,d","res     0,e",
     "res     0,h","res     0,l","res     0,(hl)","res     0,a"
```

```
4760 DATA "res     1,b","res     1,c","res     1,d","res     1,e",
     "res     1,h","res     1,l","res     1,(hl)","res     1,a"
4770 DATA "res     2,b","res     2,c","res     2,d","res     2,e",
     "res     2,h","res     2,l","res     2,(hl)","res     2,a"
4780 DATA "res     3,b","res     3,c","res     3,d","res     3,e",
     "res     3,h","res     3,l","res     3,(hl)","res     3,a"
4790 DATA "res     4,b","res     4,c","res     4,d","res     4,e",
     "res     4,h","res     4,l","res     4,(hl)","res     4,a"
4800 DATA "res     5,b","res     5,c","res     5,d","res     5,e",
     "res     5,h","res     5,l","res     5,(hl)","res     5,a"
4810 DATA "res     6,b","res     6,c","res     6,d","res     6,e",
     "res     6,h","res     6,l","res     6,(hl)","res     6,a"
4820 DATA "res     7,b","res     7,c","res     7,d","res     7,e",
     "res     7,h","res     7,l","res     7,(hl)","res     7,a"
4830 DATA "set     0,b","set     0,c","set     0,d","set     0,e",
     "set     0,h","set     0,l","set     0,(hl)","set     0,a"
4840 DATA "set     1,b","set     1,c","set     1,d","set     1,e",
     "set     1,h","set     1,l","set     1,(hl)","set     1,a"
4850 DATA "set     2,b","set     2,c","set     2,d","set     2,e",
     "set     2,h","set     2,l","set     2,(hl)","set     2,a"
4860 DATA "set     3,b","set     3,c","set     3,d","set     3,e",
     "set     3,h","set     3,l","set     3,(hl)","set     3,a"
4870 DATA "set     4,b","set     4,c","set     4,d","set     4,e",
     "set     4,h","set     4,l","set     4,(hl)","set     4,a"
4880 DATA "set     5,b","set     5,c","set     5,d","set     5,e",
     "set     5,h","set     5,l","set     5,(hl)","set     5,a"
4890 DATA "set     6,b","set     6,c","set     6,d","set     6,e",
     "set     6,h","set     6,l","set     6,(hl)","set     6,a"
4900 DATA "set     7,b","set     7,c","set     7,d","set     7,e",
     "set     7,h","set     7,l","set     7,(hl)","set     7,a"
```





TRUCS ET ASTUCES POUR L'AMSTRAD CPC

C'est le livre que tout utilisateur d'un CPC doit posséder. De nombreux domaines sont couverts (graphismes, fenêtres, langage machine) et des super programmes sont inclus dans ce best-seller (gestion de fichiers, éditeur de textes et de sons...). Réf. ML 112. 149 F. 239 p.

BEST



LE LANGAGE MACHINE DE L'AMSTRAD CPC

Ce livre est destiné à tous ceux qui désirent aller plus loin que le Basic. Des bases de programmation en Assembleur à l'utilisation des routines système, tout est expliqué avec de nombreux exemples. Contient un programme Assembleur, moniteur et désassembleur. Réf. ML 123. 129 F. 272 p.



LA BIBLE DU PROGRAMMEUR DE L'AMSTRAD CPC

Tout, absolument tout sur le CPC 464. Ce livre est l'ouvrage de référence pour tous ceux qui veulent programmer en pro leur CPC. Organisation de la mémoire, le contrôleur vidéo, les interfaces, l'interpréteur et toute la ROM désassemblée et commentée sont quelques-uns des thèmes de cet ouvrage. Réf. ML 122. 249 F.427 p.



PEEKS ET POKES DU CPC :

Comment exploiter à fond son CPC à partir du BASIC? C'est ce que vous révèle ce livre avec tout ce qu'il faut savoir sur les peeks, pokes et autres call...Vous saurez aussi comment protéger la mémoire, calculer en binaire... et tout cela très facilement . Un passage assuré et sans douleur du BASIC au puissant LANGAGE MACHINE. Réf : ML 126. 99 F. 200 p.



LE LIVRE DU LECTEUR DE DISQUETTE AMSTRAD CPC

Tout sur la programmation et la gestion des données avec le 6128 DD- 1 et le 664! Utile au débutant comme au programmeur en langage machine. Contient un listing du DOS commenté, un utilitaire qui ajoute les fichiers relatifs à l'AMDOS avec de nouvelles commandes Basic, un moniteur disque et beaucoup d'autres programmes et astuces ... Réf. ML 127 149 F. 208 p.



LE LIVRE DU CP/M PLUS

Ce livre vous permet d'utiliser CP/M sur CPC 464 et 6128 sans aucune difficulté. Vous y trouverez de nombreuses explications et les différents exemples vous assureront une maîtrise parfaite de ce très puissant système d'exploitation qu'est CP/M. Réf. ML128. 149 F 224 p.

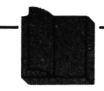


BIEN DEBUTER AVEC LE CPC 6128

Ce livre s'adresse à ceux qui débutent avec le CPC 6128. Une fois leur machine bien en main, il pourront s'attaquer au Basic et utiliser le programme de gestion d'adresses ainsi que toutes les instructions que contient ce livre.
Réf. ML 145. 99 F. 209 p.

BEST





LA BIBLE DU CPC 6128

Tout connaître sur le CPC 6128. Analyse du système d'exploitation, du processeur, le GATE ARRAY, le contrôleur vidéo, le 8255, le chip sonore, les interfaces... Comprend un désassembleur, les points d'entrée des routines commentés de l'interpréteur et du système d'exploitation. Un super livre comme toutes les bibles! Réf : ML 146. 199 F. 440 p.





LA BIBLE DU GRAPHISME

Tout sur le GSX. Ce livre est un must. Tout sur le graphisme sur CPC et PCW. Vous trouverez notamment : programmation d'un logiciel Paint, graphismes de gestion (histogrammes...), graphismes vectorisés, fonctionnement et réalisation d'un light pen, graphismes en langage machine. Et enfin, pour la première fois, des explications claires sur le GSX. Réf. ML 181. 199 F. Réf. ML 281. 299 F le livre et la disquette 550 p.



LE GRAND LIVRE DU BASIC SUR CPC 6128

Ce livre vous permet d'exploiter à fond les capacités du Basic Locomotive. Théorique et pratique, découvrez les bases de la programmation et le fonctionnement interne du Basic, et attaquez les différents domaines de la programmation : tris, fenêtres, protection, sons et musique, mémoire de masse avec l'AMDOS et le RAMDISK. Nombreux listings d'applications de haut niveau fournis et commentés. Réf. ML 166. 149 F. Réf. ML 268. 249 F. Le livre et la disquette. 265 p.



AUTOFORMATION A L'ASSEMBLEUR SUR CPC

Contient un livre et un logiciel. Ce livre permet au novice de maîtriser la programmation Z 80 grâce à la méthode efficace du Dr Watson. De nombreux exemples illustrent les différentes étapes. Des exercices (les solutions sont fournies) testent vos connaissances et peuvent être directement essayés avec le logiciel. Ce logiciel est composé d'un assembleur, d'un désassembleur et d'un programme d'exemple. Avec l'assembleur créez des programmes en langage machine pouvant être utilisés directement sous CP/M. Réf. ML 226. 195 F. (Cassette) Réf. ML 326 295 F (Disquette) Réf. ML 426 295 F (Disquette PCW) 255 p.
Déssassembleur et CP/M seulement sur PCW.

DANS LA MEME COLLECTION DES REPONSES A TOUTES VOS QUESTIONS

PROGRAMMES BASIC POUR LES CPC.
Réf. ML 118. 129 F. 164 p.

LE BASIC AU BOUT DES DOIGTS.
Réf. ML 119. 149 F. 266 p.

AMSTRAD OUVRE-TOI.
Réf. ML 120; 99 F. 205 p.

LES JEUX D'AVENTURE ET COMMENT LES PROGRAMMER.
Réf. ML 121. 129 F. 250 p.

GRAPHISMES ET SONS.
Réf. ML 124. 129 F. 185 p.

MONTAGES, EXTENSIONS ET PERIPHERIQUES DU CPC.
Réf. ML 131. 199 F. 434 p.

DES IDEES POUR LES CPC.
Réf. ML 132. 129 F. 254 p.

LES ROUTINES DE L'AMSTRAD CPC.
Réf. ML 143. 149 F. 260 p.

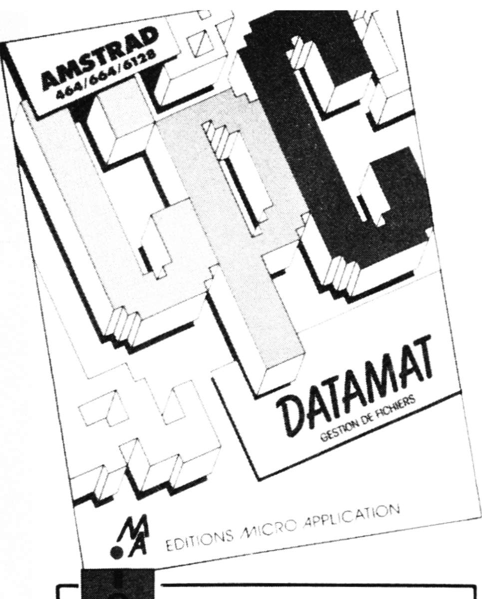
TRUCS ET ASTUCES II POUR CPC.
Réf. ML 147. 129 F. 220 p.

LE LIVRE DU LOGO.
Réf. ML 162. 149 F. 394 p.

PROGRAMMES ET APPLICATIONS EDUCATIFS SUR CPC.
Réf. ML 150. 179 F. 305 p.

COMMUNICATIONS, MODEM ET MINITEL SUR CPC.
Réf. ML 151. 149 F. 196 p.





TEXTOMAT

Un traitement de texte puissant et simple qui tire parti de toutes les capacités des CPC. Ecrivez, archivez et modifiez vos courriers, rapports, thèses, études... Intégrez dans vos documents des données extraites des fichiers DATAMAT et des calculs réalisés par CALCUMAT.

- Utilisation aisée à partir de menus.
- Jeu de caractères français complet accentué.
- Fonction de calcul en mode texte.
- Jusqu'à 16640 caractères.
- Possibilité de chainage de textes sur disquette.
- Fonctionne en mode 80 caractères avec accents.
- Travaille avec un ou deux lecteurs de disquettes.
- Choix des couleurs écran-caractères-bordure.
- Mode Insertion-Gomme...
- Tabulation.

Réf. AM 305. 390 F.





DATAMAT

DATAMAT permet de tenir à jour et d'exploiter tous vos fichiers. Deplus relié à CALCUMAT vous pourrez reprendre les données de vos fichiers pour établir des calculs et graphes (répartition géographique de vos clients, histogramme des ventes ...). Relié à TEXTOMAT vous pourrez intégrer vos données pour réaliser des mailings, courrier personnalisé...

- Emploi extrêmement simple dû à l'utilisation de menus.
- Traite tout type de menus.
- Définition d'un masque de saisie personnalisé.
- 40 à 80 caractères par ligne.
- Fonction de recopie d'écran sur imprimante.
- 50 champs par enregistrement.
- 512 caractères par enregistrement.
- Jusqu'à 4000 enregistrements par fichier.

Réf : AM 304. 390 F.

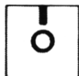


CALCUMAT

CALCUMAT est un tableur graphique de qualité professionnelle. Il se compose principalement d'une grille de calcul, d'un calepin, d'une calculatrice, d'un presse-papier et d'un module permettant la représentation graphique d'un ensemble de données. CALCUMAT s'utilise très simplement à l'aide de menus déroulants et de fenêtres de travail.

- Tri numérique ou alphanumérique d'un ensemble de cellules.
- Fonctions « couper, copier, coller » pour manipuler un ensemble de cellules par l'intermédiaire du presse papier.
- Calculs en mode automatique ou sur demande. -Représentation graphique en barres, lignes, camembert, de 4 zones de données.
- Recopie d'écran graphique sur imprimante AMSTRAD DMP 1, DMP 2000, et compatibles EPSON.
- Transfert de données de DATAMAT vers CALCUMAT pour effectuer des calculs sur les zones numériques d'un fichier.

Réf. AM 311. 390 F.



LA SOLUTION

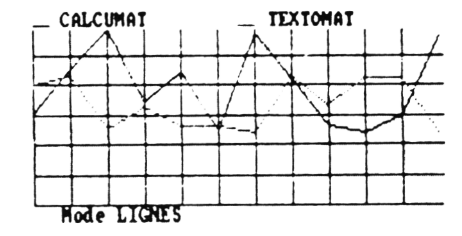
C'est votre solution BUREAUTIQUE COMPLETE sur AMSTRAD CPC. En effet, ce package regroupe trois logiciels (Traitement de Texte, Gestion de Fichiers, Tableur Graphique) complémentaires et homogènes qui vous permettent de traiter efficacement toutes vos tâches de bureau (rapport, courrier, tenue des fichiers, publipostage, calculs prévisionnels, présentation graphique des résultats...). Les trois logiciels pouvant s'échanger leurs données,les possibilités offertes par la SOLUTION sont très vastes : par exemple réalisez un mailing à partir du Traitement de Texte en reprenant les adresses sélectionnées à partir de la Gestion de Fichiers et un tableau de prévisions réalisé par le Tableur qui sera inséré dans le texte. Enfin rappelons que les trois logiciels composant LA SOLUTION sont trois best sellers internationaux de haute qualité : TEXTOMAT, DATAMAT, CALCUMAT.

Réf : AM 313. 790 F.

Mode
SECTEURS

Total

1985



Achevé d'imprimer en janvier 1988
sur les presses de l'imprimerie Laballery
58500 Clamecy
Dépôt légal : janvier 1988
Numéro d'imprimeur : 801006



GERÎTS
BRUCKMANN
ENGLISCH
STEIGERS

La Bible du CPC 6128 est un ouvrage spécialement dédié aux nouveaux AMSTRAD munis de lecteur de disquette. Ce livre est une aide indispensable pour les programmeurs en BASIC et le MUST absolu de tous les programmeurs en assembleur. Cet ouvrage de référence qui révèle vraiment tous les secrets du CPC 6128, est le fruit d'un travail minutieux de plusieurs mois.

Contenu :

I. Le HARDWARE :
- Le processeur Z80
- Le GATE ARRAY
- Le contrôleur HD6845
- La RAM et les 64K du 6128
- La RAM Vidéo
- L'interface parallèle 8255
- Le générateur de sons AY - 3 - 8912
- Le lecteur de disquette
- Les interfaces

II. Le Système d'exploitation :
- Les Vecteurs (664-6128)
- La RAM (664-6128)
- Utilisation des Routines
- Les Interruptions
- La ROM
- Le générateur de caractères

III. Le BASIC :
- L'Interpréteur (664-6128)
- La Pile
- BASIC et Langage Machine
- La ROM BASIC

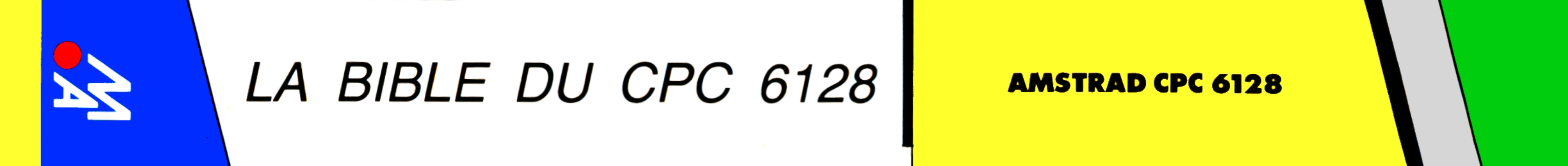
IV. ANNEXE :
- Les Routines du système d'exploitation
- Références à la RAM Système
- TOKENS BASIC
- Listing complet d'un désassembleur pour travailler sur la mémoire

ISBN : 2-86899-036-3



Réf. : ML 146   Prix : 199 FF TTC

EDITIONS MICRO APPLICATION
13, RUE SAINTE-CÉCILE 75009 PARIS
TÉL. (1) 47 70 32 44


LA BIBLE DU CPC 6128
AMSTRAD CPC 6128



